Compumotor

6000 Series

Software Reference

Compumotor Division
Parker Hannifin Corporation
p/n 88-012966-01

Parker

IMPORTANT

User Information

WARNING

Because software controls machinery, test any software control for safety under all potential operating conditions. Failure to do so can result in damage to equipment and/or serious injury to personnel.

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

North America and Asia:
Compumotor Division of Parker Hannifin
5500 Business Park Drive
Rohnert Park, CA 94928
Telephone: (800) 358-9070 or (707) 584-7558
Fax: (707) 584-3793
FaxBack: (800) 936-6939 or (707) 586-8586
BBS: (707) 584-4059
e-mail: tech_help@cmotor.com
Internet: http://www.compumotor.com

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

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

Parker
Motion & Control

Product Feedback Welcome
E-mail: 6000user@cmotor.com

Change Summary

6000 Series Software Reference

Revision K

March 1997

Revision K Changes:

- Back cover: Corrections to page number references.
- FOLEN: Corrections regarding "Restrictions on using FOLEN".
- FOLK: Description has been corrected and clarified.
- FOLSND: Correction—when you connect the step & direction device, connect Direction+ to B- on the encoder connector and connect Direction- to B+.
- TPSHF & [PSHF]: Clarification—the shift value is set to zero each time a new FOLEN1 command or a FOLMAS command (with a value other than zero) is issued.

Revision J Changes:

Topic	Description
General Information	This manual revision covers the features for these 6000 Series products with revision 4.5 firmware: **Bus-Based**: AT6200, AT6400, AT6250, AT6450 **Stand-Alone**: 610n Series, 615n Series, 6200, 6201, 6250, 6270 **OEM Level**: OEM-AT6200, OEM-AT6400, OEM6200, OEM6250 Content/Organization Changes: • The "Programming Guide" portion of previous revisions has been removed. This information is found in the *6000 Series Programmer's Guide* (p/n 88-014540-01). Overview information about command syntax, command value substitutions, programmable I/O bit patterns, and error messages is still provided in this document (see pages 1-9). • Appendix C (Command Value Substitutions) from previous revisions has been removed; follow the guidelines outlined in the *Command Value Substitutions* section on page 5. Appendix D (ASCII Table) from previous revisions has been renamed as Appendix C. • The presentation of programming examples was modified so that you can copy them from the Help system (in Motion Architect) or from the PDF file (on our www.compumotor.com web site) and paste them directly into your program (e.g., using Motion Architect's Editor).
Commanded Direction Reversal (CMDDIR)	**Enhancement**: The commanded direction polarity reversal command (CMDDIR) is available for the 615n series, the 6270, and all stepper products (610n, AT6n00, 620n). 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 or motor and the feedback device, or (b) change the sign of motion-related commands in your program.

Topic	Description
Compiled Motion	**New Feature**: (see *6000 Series Programmer's Guide* for detailed description) Related commands (new): FOLRNF .... Numerator of Final Slave-to-Master Ratio, Preset Moves GOBUF ..... Store a Motion Segment in Compiled Memory PLN ..... Loop End, compiled motion PLOOP ..... Loop Start, compiled motion POUTA ..... Output on Axis 1, compiled motion POUTB ..... Output on Axis 2, compiled motion POUTC ..... Output on Axis 3, compiled motion POUTD ..... Output on Axis 4, compiled motion [SEG] .... Number of Segments Available In Compiled Memory TSEG..... Transfer Number of Segments Available, Compiled Memory VF ..... Final Velocity Existing commands, modified to support compiled motion: GOWHEN.....Conditional GOs allowed in compiled motion PCOMP .....Pre-Compile a Program PRUN.....Run a Pre-Compiled Program PUCOMP.....Un-Compile a Program [SS].....Bit #29 set if compiled memory 75% full, bit #30 set if 100% full Bit #31 is set if a compile (PCOMP) failed; cleared on power-up, reset, or after a successful compile. (See *Status Reporting, Additions* below for a list of typical causes.) TRGFN .....Execute GOWHENs or start new master cycle in compiled motion TSS.....(see [SS] description above)
Contouring (Circular Interpolation)	**Enhancement**: As of Rev 4.1, the Contouring feature is now available for <u>all</u> multi-axis products, steppers and servos. (see *Path Contouring* command group on inside back cover)
Continuous Command Execution Mode (COMEXC1)	**Enhancement**: On-The-Fly changes (pre-emptive motion). In addition to velocity (V), acceleration (A & AA), and deceleration (AD & ADA), you may now change the positioning mode (MC & MA), the distance (D), and the Following ratios (FOLRN & FOLRD). These changes will affect the subsequent GO command executed while moving; thus, this new enhancement is referred to as "pre-emptive GOs." When pre-processing subsequent moves, the subsequent move may now be executed as soon as the next GO command is executed. Previous to revision 4.0, the subsequent move could not be executed until all moves on all axes were completed.
Drive Configuration & Reset	**Enhancements**: New commands added to set up the drive component of the 610n: DACTDP.... Enable/disable active damping for speeds greater than 3 rps. (configuration procedure: see the *ZETA6104 Installation Guide*) DAREN ..... Enable/disable anti-resonance. Anti-resonance is inhibited at or below 3 rps, and if active damping is enabled. DELVIS.... Enable/disable electronic viscosity for speeds at or below 3 rps. (configuration procedure: see the *ZETA6104 Installation Guide*) DAUTOS.... Enable/disable automatic current standby mode in which current to the motor (<u>& torque</u>) is reduced by 50% if no pulses are sent for 1 second. Full current is restored upon the next pulse. DMTIND.... Motor inductance (used only for active damping—DACTDP). DMTSTT.... Motor static torque (used only for active damping—DACTDP). DWAVEF.... Motor waveform (required for matching the motor to the drive). 615n only: As of Rev 4.1, you may use the new DRESET command to reset the internal drive independent of the internal controller. The purpose of the DRESET command is to clear fault conditions with the internal drive.
Encoder Polarity Reversal (ENCPOL)	**Enhancement**: The encoder polarity reversal command (ENCPOL) is now available to all 6000 stepper products (AT6n00, 620n, & 610n). Previous to 4.0 the ENCPOL command was only applicable to the 6270. The ENCPOL command is used to reverse the polarity (counting direction) of the encoder feedback counts. This is an alternative to reversing the A+ and A- connections to the encoder.

Topic	Description
Error Checking Conditions	**Enhancements**: • 610n: The *drive fault* error (reported with error status bit #4 and axis status bit #14) can be caused by any one or combination of the factors list below. To ascertain the exact cause, use the extended axis status (see TASX, TASXF, or ASX): - Motor fault (disconnected/faulty motor cable or short in motor) — bit #1 - Low-voltage (power) — bit #2 - Maximum drive temperature (131°F, 55°C) exceeded — bit #3 • Error status enhancements (see [ER], TER, TERF, ERROR, and ERRORP): - Error bit #8 is set if a stop input (assigned with INFNCi-D) is activated. - Error bit #10 is set if the target position specified for a pre-emptive GO or a registration move is not achievable at the time the pre-emptive GO command is executed or the registration input is activated. This condition also sets bit #30 in the axis status register (reported with TAS & AS). To clear error bit #10 and axis bit #30, execute another GO command. • Related commands: [ER] ..... Error Status (assignment or comparison) ERROR..... Error-Checking Enable ERRORP .... Error Program Assignment TER ..... Transfer Error Status (see also TERF)
Fast Status (bus-based products)	**Correction**: The bit assignments for the Limits status in block 5 are not the same as those for the TLIM report. A new table of Limit Bit Assignments has been added to the FASTAT command description. **Clarification**: The input buffer is 256 bytes.
Following	**Enhancements**: • The new Following Kill (FOLK) command allows you to limit what will kill the Following profile. That is, it allows the slave to remain in synchronization with the master even after the occurrence of a drive fault, user fault input, excess position error, or enable or pulse-cut input. • The new Numerator of Final Slave-to-Master Ratio, Preset Moves (FOLRNF) command designates that the motor will move the load the distance assigned in the preset GOBUF segment, completing the move at a final ratio of zero. FOLRNF applies only to the first subsequent GOBUF, which marks an inter-mediate "end of move" within a Following profile. The FOLRNF command is only useful for compiled Following moves. • The new Following Step and Direction (FOLSND) command allows you to Follow the step & direction input connected to the encoder connector (this is an alternative to the standard quadrature encoder input).
Homing	**Clarification**: Avoid using pause and resume functions during the homing operation (HOM). A pause command (PS or !PS) or pause input (input configured with the INFNCi-E command) will pause the homing motion. However, when the subsequent resume command (C or !C) or resume input (INFNCi-E input) occurs, motion will resume at the beginning of the homing motion sequence.
Memory Management	**Enhancements**: • Compiled Memory status commands: - System status (TSS, TSSF, & SS) bit #29 is set if compiled memory is 75% full, bit #30 is set if compiled memory is 100% full - TSEG & SEG report the number of available segments in compiled memory • All stand-alone products are shipped with 150,000 bytes of memory. The -M option has thus been eliminated for these products. • The second field in the MEMORY command is re-defined to be for "compiled memory" (i.e., anything compiled with the PCOMP command). • These commands are automatically saved in non-volatile memory: CMDDIR .... Commanded Direction Polarity (610n, 615n, 620n, 6270 only) DMTIND .... Motor Inductance (610n only) DMTSTT .... Motor Static Torque (610n only) DRPCHK .... RP240 check (610n, 615n, 620n, & 625n only) ENCPOL .... Encoder Polarity (610n, 620n, & 6270 only)

Topic	Description
On-The-Fly Motion (AKA: *Pre-Emptive GOs*)	**Enhancements**: (see GO command description) • The two basic ways of creating a complex profile are with compiled buffered motion, or with pre-emptive GOs. With compiled buffered motion, portions of a profile are built piece by piece, and stored for later execution. Compiled buffered motion is appropriate for motion profiles with motion segments of pre-determined velocity, acceleration and distance. With pre-emptive GOs, the motion profile underway is pre-empted with a new profile when a new GO is issued. The new GO both constructs and launches the pre-empting profile. Pre-emptive GOs are appropriate when the desired motion parameters are not known until motion is already underway. • Affected Commands: COMEXC .... COMEXC1 mode allows pre-emptive motion with buffered commands GO ..... Allows pre-emptive D, MC, MA, FOLRN, & FOLRD changes TAS & AS ... Bit #30 is set if the load has already passed the target position (D) specified in a pre-emptive GO. (also sets error status bit #10) TER & ER ... Error status bit #10 is set if axis status bit #30 is set.
Registration	**Enhancements**: • New Commands: REGLOD .... Registration Lock-Out Distance. Establishes a *lock-out* distance (measured from the start of motion to the current actual position) to be traveled before a registration move is allowed. REGSS ..... Registration Single-Shot. Allows only one registration move on the specified axis. Prevents other triggers from interrupting the registration move in progress. • Axis status bit #28, reported by the TAS, TASF and AS commands, is set to 1 when a registration move has been initiated by any registration input (trigger). Bit #28 is cleared (set to 0) upon the next GO command for that axis. • If, when the registration input is activated, the registration move profile cannot be performed with the specified parameters, the 6000 controller will kill the move in progress and set axis status bit #30 (see TAS & AS). If error-checking bit #10 is enabled with the ERROR command, the controller will also set error status bit #10 (see TER, TERF & ER) and branch to the assigned ERRORP error-handling program. Axis status bit #30 and error status bit #10 are cleared (set to 0) upon the next GO command for that axis. • As of revision 4.1, Registration is now available all 6000 products (previous to 4.1, Registration was available only for stepper products).
Serial Communication	**Enhancements**: • BOT command was created to control the beginning-of-transmission characters for all responses from the 6000 product. • XONOFF command (new) enables/disables XON/XOFF ASCII handshaking. • Additional features to control multiple serial ports on stand-alone products: [.....Send response from the subsequent command to both ports.] .....Send response from the subsequent command to the alternate port from the one selected with the most recent PORT command. DRPCHK .....Configures the serial port (specified with the last PORT command) to be used with an RP240, or 6000 commands, or both. PORT .....Determines which serial port is affected by the subsequent DRPCHK, E, ECHO, BOT, EOT, EOL, ERRORK, ERRBAD, ERRDEF, ERRLVL, and XONOFF commands. *Continued on next page*

<table>
<tr><th>Topic</th><th>Description</th></tr>
<tr><td>Serial Communication
(continued from previous page)</td><td>
• As of 4.0, the ECHO command was enhanced with options 2 and 3. The purpose is to accommodate an RS-485 multi-drop configuration in which a host computer communicates to the “master” 6000 controller over RS-232 (COM1 port) and the master 6000 controller communicates over RS-485 (COM2 port) to the rest of the units on the multi-drop. For this configuration, the echo setup should be configured by sending to the master the following commands executed in the order shown. In this example, it is assumed that the master's device address is set to 1. Hence, each command is prefixed with “1_” to address only the master unit.
1_PORT2.. Subsequent command affects COM2, the RS-485 port
1_ECHO2.. Echo characters back through the other port, COM1
1_PORT1.. Subsequent command affects COM1, the RS-232 port
1_ECHO3.. Echo characters back through both ports, COM1 and COM2
</td></tr>
<tr><td>Servo Output, ANA (AT6n50 only)</td><td>
Enhancement:
Use the OUTANA command to control the analog output voltage on the AT6n50's ANA output terminal.
</td></tr>
<tr><td>Servo Updates Changed
(Servo Products Only)</td><td>
(see SSFR command description for full explanation of table contents)
<table>
<tr><th></th><th></th><th colspan="2">Servo Sampling Update</th><th colspan="2">Motion Trajectory Update</th><th colspan="2">System Update</th></tr>
<tr><th># of Axes (INDAX)</th><th>SSFR Setting</th><th>Frequency (samples/sec.)</th><th>Period (µsec)</th><th>Frequency (samples/sec.)</th><th>Period (µsec)</th><th>Frequency (samples/sec.)</th><th>Period (µsec)</th></tr>
<tr><td>1</td><td>1</td><td>3030</td><td>330</td><td>3030</td><td>330</td><td>757</td><td>1320</td></tr>
<tr><td>1</td><td>2</td><td>5405</td><td>185</td><td>2703</td><td>370</td><td>675</td><td>1480</td></tr>
<tr><td>Default, Single-Axis 1</td><td>4</td><td>6250</td><td>160</td><td>1563</td><td>640</td><td>520</td><td>1920</td></tr>
<tr><td>1</td><td>8</td><td>6667</td><td>150</td><td>833</td><td>1200</td><td>417</td><td>2400</td></tr>
<tr><td>2</td><td>1</td><td>2353</td><td>425</td><td>2352</td><td>425</td><td>588</td><td>1700</td></tr>
<tr><td>2</td><td>2</td><td>3571</td><td>280</td><td>1786</td><td>560</td><td>446</td><td>2400</td></tr>
<tr><td>Default, Two-Axis 2</td><td>4</td><td>3571</td><td>280</td><td>893</td><td>1120</td><td>446</td><td>2400</td></tr>
<tr><td>2</td><td>8</td><td>3571</td><td>280</td><td>446</td><td>2240</td><td>446</td><td>2400</td></tr>
<tr><td>3</td><td>1</td><td>1667</td><td>600</td><td>1667</td><td>600</td><td>555</td><td>1800</td></tr>
<tr><td>3</td><td>2</td><td>2222</td><td>450</td><td>1111</td><td>900</td><td>555</td><td>1800</td></tr>
<tr><td>3</td><td>4</td><td>2353</td><td>425</td><td>588</td><td>1700</td><td>588</td><td>1700</td></tr>
<tr><td>4</td><td>1</td><td>1250</td><td>800</td><td>1250</td><td>800</td><td>417</td><td>2400</td></tr>
<tr><td>4</td><td>2</td><td>1667</td><td>600</td><td>833</td><td>1200</td><td>417</td><td>2400</td></tr>
<tr><td>Default, Four-Axis 4</td><td>4</td><td>2000</td><td>500</td><td>500</td><td>2000</td><td>500</td><td>2000</td></tr>
</table>
</td></tr>
<tr><td>Status Reporting</td><td>
Enhancements:
• New transfer (display status) commands:
TASX ..... Transfer extended axis status (see also TASXF). Bit assignments:
Bit #1: Motor fault (610n only)
Bit #2: Drive low voltage fault (610n only)
Bit #3: Drive over-temperature fault (610n only)
Bit #4: Drive fault input is active
TSEG ..... Transfer number of segments available in compiled memory
• New assignment/comparison operators:
SEG ..... Number of segments available in compiled memory
ASX ..... Extended axis status information
• Pre-emptive Motion and Registration status:
TAS & AS .... Axis status bit #28 is set if a registration move occurs.
Bit #30 is set if the profile specified for a pre-emptive GO or registration move is not possible at the time of the GO or the registration input (also sets error status bit #10).
TER & ER..... Error status bit #8 is set if a stop input (INFNCi-D) is activated.
Bit #10 is set if axis status bit #30 is set.
Bit #16 is set if a bad command is detected; cleared with TCMDER.

Continued on next page
</td></tr>
</table>

Topic	Description
Status Reporting *(continued from previous page)*	• Compiled profile status: TSS & SS ..... System status bit #29 is set if compiled memory is 75% full. Bit #30 is set if compiled memory is 100% full. Bit #31 is set if a compile (PCOMP) failed, cleared on power-up, reset, or after a successful compile. Possible causes include: - Errors in profile design (e.g., change direction while at non-zero velocity, distance & velocity equate to < 1 count/system update, preset move profile ends in non-zero velocity) - Profile will cause a Following error (see TFS & FS status) - Out of memory (see system status bit #30) - Axis already in motion at the time of the PCOMP command - Loop programming errors (e.g., no matching PLOOP or PLN, more than 4 embedded PLOOP/END loops) TSEG & SEG. Report number of available segments in compiled memory. • Drive Fault Input Status: As of revision 4.1, extended axis status (TASX & ASX) bit #4 is now available to check the drive fault input status whether or not the drive is enabled (DRIVE1) or disabled (DRIVEØ). Previous to revision 4.1, the status of the drive fault input could only be checked while the drive was enabled (DRIVE1) and was reported only with axis status (TAS & AS) bit #14 and error status (TER & ER) bit #4. The branch to the error program has not been changed—the error program is called only if the drive fault occurs while the drive is enabled. • The INDUST command (which allows you to create your own custom status word based on other status registers) now allows you to use the status bits from the extended axis status (see TASX description above). In the syntax INDUSTi-ic, the options for "c" (the status register source) now include L, M, N and O, representing the extended axis status registers for axes 1, 2, 3 and 4, respectively. For additional details on creating a custom user status word, refer to the INDUST command description. • As of Rev 4.1, the TVELA command is now applicable to all stepper controllers using encoder feedback (previously only for servos). For steppers, the TVELA command reports the current velocity (in revs/sec) as derived from the encoder. The reported value is <u>not</u> affected by scaling. The VELA assignment/comparison operator for TVELA is now available as of rev 4.0. **Clarification**: Some status commands also have a full-text version that takes the guess-work out of identifying the function of each status bit in a binary report. TASF..... Axis Status TASXF ..... Axis Status, Extended TERF..... Error Status TFSF..... Following Status TINOF ..... Other Inputs Status (joystick inputs, and the P-CUT or ENBL input) TSSF..... System Status
Target Zone	**Enhancement**: • The Target Zone mode allows you to define what the controller considers a "completed move," based on specified end-of-move distance, velocity, and settling time parameters. As of revision 4.0, the Target Zone mode is now applicable to <u>all</u> 6000 products (previous to 4.0, the Target Zone mode was available only for servo products). NOTE: Steppers require encoder feedback (and ENC1 mode) for this feature. • Target Zone Commands: STRGTE.... Target Zone Mode Enable/Disable STRGTD.... Target Distance Zone STRGTT.... Target Settling Timeout Period STRGTV.... Target Velocity Zone
Variables, Clearing	**Enhancement**: You can clear all numeric (VAR), binary (VARB) and string (VARS) variables at one time by using the VARCLR command (sets all variables to factory default values).
Velocity (actual) Assignment	**Enhancement**: Using the new [VELA] operator, you can assign/compare (e.g., for a conditional expression) the current velocity as measured from the feedback device. Use [VEL] for commanded velocity, and [V] for programmed velocity.

Introduction

Purpose of this Document

This document is designed as a reference for all the 6000 Series commands. To gain a full understanding of how the 6000 Series commands are used together to implement specific features, refer to the *6000 Series Programmer's Guide* (p/n 88-014540-01). For hardware-related information (e.g., electrical wiring connections, specifications, tuning, etc.), refer to your product's *Installation Guide*.

Table of Contents

Description of Format

	1.	2.	3.
	INEN	**Input Enable**	
4.	Type	`Inputs or Program Debug Tools`	
5.	Syntax	`<!>INEN<d><d><d>...<d>`	
6.	Units	`d = 0, 1, E, or X`	
7.	Range	`0 = off, 1 = on, E = enable, X = don't care`	
8.	Default	`E`	
9.	Response	`INEN: *INENEEEE_EEEE_EEEE_EEEE_EEEE_EEEE_EEEE_EEEE`	
10.	See Also	`[IN], INFEN, INFNC, INLVL, INPLC, INSTW, TIN`	

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	1.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

Item Number	Description
1.	**Mnemonic Code**: This field contains the command's mnemonic code. If the command is in brackets (e.g., `[ IN ]`), it is an operator that must be used within the syntax of another command (e.g., `IN` may be used in a conditional expression like `IF(IN.12=b1)`).
2.	**Full Name**: This field contains the command's full name.
3.	**Valid Product & Revision**: This field lists the 6000 Series products and the revision of each product when this command was incorporated or modified per the description. If the command does not apply to that particular product, the **Rev** is specified as “n/a”. All commands applicable to the standard product versions are applicable to the OEM versions unless otherwise noted (e.g., 6250 commands are applicable to the OEM6250 controller). An “n” in the product name refers to all products in that particular series (e.g., 620n commands are applicable to the 6200 and 6201 products), and the commands are applicable to all members of the series unless otherwise noted. You can use the `TREV` command to determine which product revision you are using. For example, if the `TREV` response is `*TREV92-012222-01-4.1`, the product revision is 4.1.
4.	**Type**: This field contains the command's type. Inside the back cover you will find a list of all 6000 Series commands organized by command type.
5.	**Syntax**: The proper syntax for the command is shown here. The specific parameters associated with the command are also shown. Definitions of the parameters are described in the *Syntax* sections below.
6.	**Units**: This field describes what unit of measurement the parameter (`b`, `d`, `i`, `r`, or `t`) in the command syntax represents.
7.	**Range**: This is the range of valid values that you can specify for an argument (or any other parameter specified).
8.	**Default**: The default setting for the command is shown in this field. A command will perform its function with the default setting if you do not provide a value.
9.	**Response**: Some commands allow you to check the status of the command. In the example above, entering the `INEN` command by itself, you will receive the response `*INENEEEE_EEEE_EEEE_EEEE_EEEE_EEEE_EEEE_EEEE` (response indicates all inputs are enabled). The example responses provided are based on the default error level, Error Level 4, established with the `ERRLVL4` command.
10.	**See Also**: Commands related or similar to the command described are listed here.

Syntax -- Letters and Symbols

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

> **NOTE**
>
> For more detailed information on 6000 command syntax, refer to the *6000 Series Programmer's Guide*.

INEN — Input Enable

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

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	1.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

Letter/Symbol	Description
a	Represents an axis specifier, numeric value from 1 to 4 (used only to elicit a response from the indexer)
b*	Represents the values 1,0, X or x; does not require field separator between values.
c	Represents a character (A to Z, or a to z)
d	Represents the values 1,0, X or x, E or e ; does not require field separator between values. E or e enables a specific command field. X or x leaves the specific command field unchanged or ignored.
i	Represents a numeric value that cannot contain a decimal point (integer values only). The numeric range varies by command. Field separator required.
r	Represents a numeric value that may contain a decimal point, but is not required to have a decimal point. The numeric range varies by command. Field separator required.
t	Represents a string of alpha numeric characters from 1 to 6 characters in length. The string must start with a alpha character.
!	Represents an immediate command. Changes a buffered command to an immediate command. Immediate commands are processed immediately, even before previously entered buffered commands.
,	Represents a field separator. Commands with the symbol r or i in their Syntax description require field separators. Commands with the symbol b or d in their Syntax description **do not** require field separators (but they may be included). See *General Guidelines* table below.
@	Represents a global specifier, where only one field need be entered. Applicable to all commands with multiple command fields. (e.g., @V1 sets velocity on all axes to 1 rps)
< >	Indicates that the item contained within the < > is optional, not required by that command. NOTE: Do not confuse with <cr>, <sp>, and <lf>, which refer to the ASCII characters corresponding to a carriage return, space, and line feed, respectively.
[ ]	Indicates that the command between the [ ] must be used in conjunction with another command, and cannot be used by itself.

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

Syntax -- General Guidelines

Guideline Topic	Guideline	Examples
Neutral Characters (<sp> and <tab>)>)	Using neutral characters anywhere within a command will not affect the command.	Set velocity on axis 1 to 10 rps and axis 2 to 25 rps: V<sp>10,<sp>25,,<cr> Add a comment to the command: V 10, 25,,<tab> ;set accel.<cr>
Case Sensitivity	There is no case sensitivity. Use upper or lower case letters within commands.	Initiate motion on axes 1, 3 and 4: GO1011<cr> go1011<cr>
Command Delimiters (<cr>, <lf>, and :)	All commands must be separated by a command delimiter.	Set acceleration on axis 2 to 10 rev/sec/sec: A,10,,<cr> A,10,,<lf> A,10,,:
Comment Delimiter (;)	All text between a comment delimiter and a command delimiter is considered *program comments*.	Add a comment to the command: V10<tab> ;set velocity<cr>
Field Separator (,)	Commands with the symbol r or i in their Syntax description require field separators. Commands with the symbol b or d in their Syntax description **do not** require field separators (but they may be included). Axes not participating in the command need not be specified; however, field separators that are normally required must be specified.	Set velocity on axes 1 - 4 to 10 rps, 25 rps, 5 rps and 10 rps, respectively: V10,25,5,10<cr> Initiate motion on axes 1, 3 and 4: GO1011<cr> GO1,0,1,1<cr> Set velocity on axis 2 to 5 rps: V,5,,<cr>
Global Command Identifier (@)	When you wish to set the command value equal on all axes, add the @ symbol at the beginning of the command (enter only the value for one command field).	Set velocity on all axes to 10 rps: @V10<cr>
Bit Select Operator (.)	The bit select operator allows you to affect one binary bit without having to enter all the preceding bits in the command. Syntax for setup commands: [command name].[bit #]-[binary value] Syntax for conditional expressions: [command name].[bit #]=[binary value]	Enable error-checking bit #9: ERROR.9-1<cr> IF statement based on value of axis status bit #12: IF(1AS.12=b1)<cr>
Left-to-right Math	All mathematical operations assume left-to-right precedence.	VAR1=5+3*2<cr> Result: Variable 1 is assigned the value of 16 (8*2), not 11 (5+6).

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

Syntax -- Command Value Substitutions

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

VARB..... Uses the value of the binary variable to establish all the fields in the command.

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

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

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

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

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

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

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

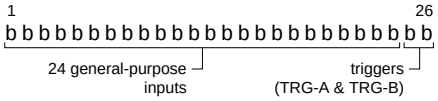
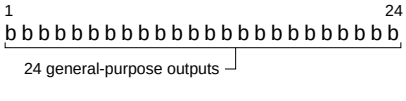
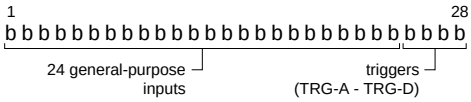
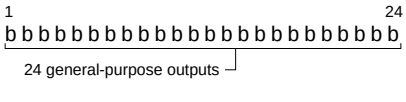
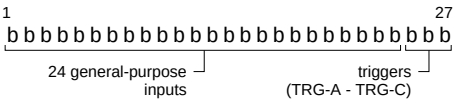
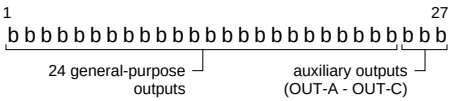
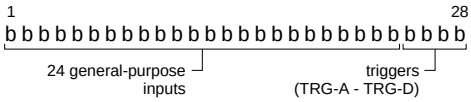
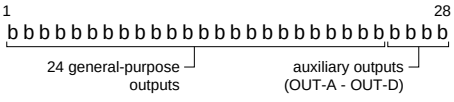
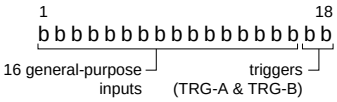
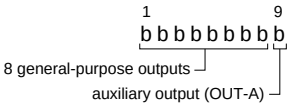
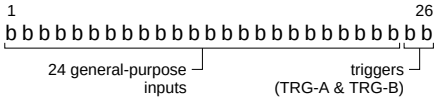
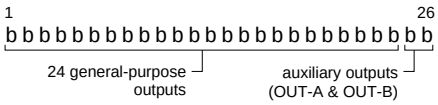
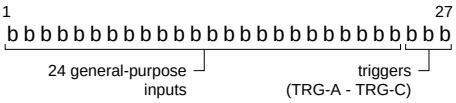
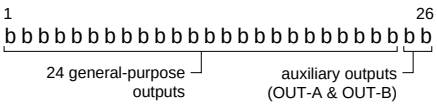
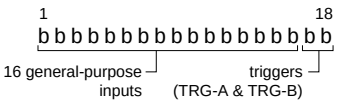
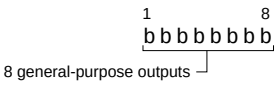
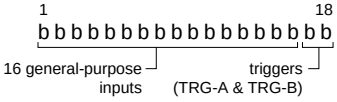
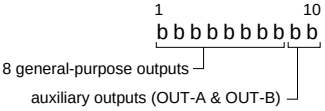
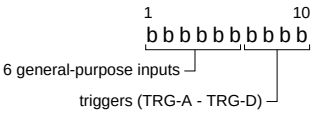
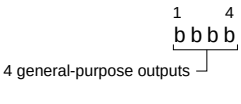
```
VAR1=15              ; Set variable 1 to 15
A5,(VAR1),4,4        ; Set acceleration to 5,15,4,4 for axes 1-4, respectively
VARB1=b1101XX1       ; Set binary variable 1 to 1101XX1 (bits 5 & 6 not affected)
GO(VARB1)            ; Initiate motion on axes 1, 2 & 4 (value of binary
                  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

Rule of Thumb

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

Programmable I/O Bit Patterns

The total number of programmable inputs and outputs, including trigger inputs and auxiliary outputs, varies from one 6000 Series product to another. Consequently, the bit patterns for the programmable inputs and outputs also vary by product. For example, for the AT6400 the TRG-A trigger input is represented by programmable input bit #25, but for the ZETA6104 the TRG-A trigger input is bit #17. Bit numbers are referenced in commands like `WAIT(IN.13=b1)`, which means wait until programmable input #13 becomes active. To ascertain your product's I/O bit patterns, refer to table below. I/O pin outs, specifications, and circuit drawings are provided in your product's *Installation Guide*.

Product	Programmable Input Pattern	Programmable Output Pattern
AT6200		
AT6400		
AT6250		
AT6450		
610n Series, 615n Series		
620n Series, 6270		
625n Series		
OEM6200		
OEM6250		
OEM-AT6400		

Programming Error Messages

Depending on the error level setting (set with the ERRLVL command), when a programming error is created, the 6000 controller will respond with an error message and/or an error prompt. A list of all possible error messages is provided in a table below. The default error prompt is a question mark (?), but you can change it with the ERRBAD command if you wish.

At error level 4 (ERRLVL4—the factory default setting) the 6000 controller responds with both the error message and the error prompt. At error level 3 (ERRLVL3), the 6000 controller responds with only the error prompt.

Error Response	Possible Cause
ACCESS DENIED	Program security feature enabled, but program access input (INFNCi-Q) not activated
ALREADY DEFINED FOR THUMBWHEELS	Attempting to assign an I/O function to an I/O that is already defined as a thumbwheel I/O
AXES NOT READY	Compiled Profile path compilation error
COMMAND NOT IMPLEMENTED	Command is not applicable to the 6000 Series product
ERROR: MOTION ENDS IN NON-ZERO VELOCITY - AXIS *N*	Compiled Motion: The last GOBUF segment within a PLOOP/PLN loop does not end at zero velocity, or there is no final GOBUF segment placed outside the loop.
EXCESSIVE PATH RADIUS DIFFERENCE	Contouring path compilation error
FOLMAS NOT SPECIFIED	No FOLMAS for the axis is currently specified. It will occur if FMCNEW, FSHFC, or FSHFD commands are executed and no FOLMASØ command was executed, or FOLMAS0 was executed.
INCORRECT AXIS	Axis specified is incorrect
INCORRECT DATA	Incorrect command syntax. Following: Velocity (V), acceleration (A) or deceleration (AD) command is zero (used by FSHFC & FSHFD).
INSUFFICIENT MEMORY	Not enough memory for the user program or compiled profile segments. This may be remedied by reallocating memory (see MEMORY command description).
INVALID COMMAND	Command is invalid because of existing conditions
INVALID CONDITIONS FOR COMMAND	System not ready for command (e.g., LN command issued before the L command). Following (these conditions can cause an error during Following): • The FOLMD value is too small to achieve the preset distance and still remain within the FOLRN/FOLRD ratio. • A phase shift cannot be performed: FSHFD .... Error if already shifting or performing other time based move. FSHFC .... Error if currently executing a FSHFD move, or if currently executing another FSHFC move in the opposite direction. • The FOLEN1 command was given while a profile was suspended by a GOHWHEN.
INVALID CONDITIONS FOR S_CURVE ACCELERATION–FIELD *n*	Average (Aavg) acceleration or deceleration command (e.g., AA, ADA, HOMAA, HOMADA, etc.) with a range that violates the equation ½Amax ≤ Aavg ≤ Amax (Amax is the max. accel or decel command—e.g., A, AD, HOMA, HOMAD, etc.)

Programming Error Messages *(continued)*

Error Response	Possible Cause
INVALID DATA	Data for a command is out of range. Following (these conditions can cause an error during Following): • The parameter supplied with the command is valid. FFILT.....Error if: smooth number is not 0-4 FMCLEN...Error if: master steps > 999999999 or negative FMCP.....Error if: master steps > 999999999 or < -999999999 FOLMD.....Error if: master steps > 999999999 or negative FOLRD.....Error if: master steps > 999999999 or negative FOLRN.....Error if: slave steps > 999999999 or negative FSHFC.....Error if: number is not 0-3 FSHFD.....Error if: slave steps > 999999999 or < -999999999 GOWHEN...Error if: position > 999999999 or < -999999999 WAIT.....Error if: position > 999999999 or < -999999999 • Error if a GO command is given in the preset positioning mode (MCØ) and: FOLRN = zero FOLMD = zero, or too small (see Following chapter in the *6000 Series Programmer's Guide*)
INVALID FOLMAS SPECIFIED	Following: An illegal master was specified in FOLMAS. A slave may never use its own commanded position or feedback source as its master.
INVALID RATIO	Following: Error if the FOLRN:FOLRD ratio after scaling is > 127 when a GO is executed
LABEL ALREADY DEFINED	Defining a program or label with an existing program name or label name
MAXIMUM COMMAND LENGTH EXCEEDED	Command exceeds the maximum number of characters
MAXIMUM COUNTS PER SECOND EXCEEDED	Velocity value is greater than 1,600,000 counts/sec
MOTION IN PROGRESS	Attempting to execute a command not allowed during motion (see Restricted Commands During Motion section in the *6000 Series Programmer's Guide*.) Following: The FOLEN1 command was given while that slave was moving in a non-Following mode.
NEST LEVEL TOO DEEP	IFs, REPEATs, WHILEs, & GOSUBs nested greater than 16 levels
NO MOTION IN PROGRESS	Attempting to execute a command that requires motion, but motion is not in progress
NO PATH SEGMENTS DEFINED	Compiled Profile compilation error
NO PROGRAM BEING DEFINED	END command issued before a DEF command
NOT ALLOWED IF SFBØ	Changes to tuning commands (SGILIM, SGAF, SGAFN, SGI, SGIN, SGP, SGPN, SGV, SGVN, SGVF, and SGVFN) and SMPER are not allowed if SFBØ is selected
NOT ALLOWED IN PATH	Compiled Profile path compilation error
NOT DEFINING A PATH	Executing a compiled profile or contouring path command while not in a path
NOT VALID DURING FOLLOWING MOTION	A GO command was given while moving in the Following mode (FOLEN1) and while in the preset positioning mode (MCØ).
NOT VALID DURING RAMP	A GO command was given while moving in a Following ramp and while in the continuous positioning mode (MC1). Following status (FS) bit #3 will be set to 1. A FOLEN command was given during one of these conditions: • During a shift (FSHFC or FSHFD) • During a change in ratio (FOLRN/FOLRD) • During deceleration to a stop
PATH ALREADY MOVING	Compiled Profile path compilation error
PATH NOT COMPILED	Attempting to execute a individual axis profile or a multiple axis contouring path that has not been compiled

Programming Error Messages *(continued)*

Error Response	Possible Cause
PATH RADIUS TOO SMALL	Contouring path compilation error
PATH RADIUS ZERO	Contouring path compilation error
PATH VELOCITY ZERO	Contouring path compilation error
STRING ALREADY DEFINED	A string (program name or label) with the specified name already exists
STRING IS A COMMAND	Defining a program or label that is a command or a variant of a command
UNDEFINED LABEL	Command issued to product is not a command or program name
WARNING: POINTER HAS WRAPPED AROUND TO DATA POINT 1	During the process of writing data (DATTCH) or recalling data (DAT), the pointer reached the last data element in the program and automatically wrapped around to the first datum in the program
WARNING: ENABLE INPUT INACTIVE	Servo controllers only: **ENBL** input is no longer connected to ground (**GND**)
WARNING: PULSE CUT INPUT ACTIVE	Stepper controllers only: **P-CUT** input is no longer connected to ground (**GND**)
WARNING: DEFINED WITH ANOTHER TW/PLC	Duplicate I/O in multiple thumbwheel definitions

Identifying Bad Commands

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

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

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

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

Example Error Scenario:

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

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

2. Using Motion Architect's terminal emulator, download the program to the 6000 Series product. Notice that an error response identifies the bad command as an "INCORRECT DATA" item and displays it:

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

[!] Immediate Command Identifier

Type	Operator (Other)
Syntax	!<command>
Units	n/a
Range	n/a
Default	n/a
Response	n/a
See Also	COMEXC

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

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

All 6000 Series commands are buffered.

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

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

[@] Global Command Identifier

Type	Operator (Other)
Syntax	@<command><field1>
Units	n/a
Range	n/a
Default	n/a
Response	n/a
See Also	INDAX

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

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

; Begin Comment

Type	Operator (Other)
Syntax	;<this is a comment>
Units	n/a
Range	n/a
Default	n/a
Response	n/a
See Also	None

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

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

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

$ Label Declaration

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

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

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

Maximum number of labels possible:

All stand-alone products	600
AT6n00	250
AT6n00-M	6000
AT6n50	100
AT6n50-M	600

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

NOTE: A label cannot have the same name as a 6000 Series command. For example, $A and $A123 are illegal labels.

Example

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

[#] Step Through a Program

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

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

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

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

Example:

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

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

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

!

Enter Interactive Data

		Product	Rev
Type	Operator (Other)		
Syntax	!'<numeric data>	AT6n00	1.0
Units	Numeric data is command-dependent	AT6n50	1.0
Range	Numeric data is command-dependent	610n	4.0
Default	n/a	615n	1.0
Response	n/a	620n	1.0
See Also	[READ], VARS	625n	1.0
		6270	1.0

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

Command processing will pause while waiting for data.

Example:

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

[.] Bit Select

		Product	Rev
Type	Operator (Other)	AT6n00	1.0
Syntax	<command>.i	AT6n50	1.0
Units	i = bit number	610n	4.0
Range	Command-dependent	615n	1.0
Default	None	620n	1.0
Response	n/a	625n	1.0
See Also	[AS], [ER], ERROR, [IN], INEN, INLVL, [INO], INTHW, LHLVL, [LIM], [MOV], ONIN, ONUS, OUT, OUTEN, OUTLVL, POUT, [SS], TAS, TER, TIN, TINO, TINT, TLIM, TOUT, TSS, TUS, [US]	6270	1.0

The Bit Select (.) command specifies which bit of an assignment command or a transfer command to select. The primary purpose of this command is to let the user specify a specific bit, instead of a bit string.

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

Example:

```
VARB2=ER.12     ; Error status bit 12 assigned to binary variable 2
VARB2           ; Response (if bit 12 is set to 1):
                ; *VARB2=XXXX_XXXX_XXX1_XXXX_XXXX_XXXX_XXXX_XXXX
OUT.5-1         ; Turn output bit #5 on
```

["] Begin and End String

		Product	Rev
Type	Operator (Other)	AT6n00	1.0
Syntax	"<message>" (see below for possibilities)	AT6n50	1.0
Units	n/a	610n	4.0
Range	n/a	615n	1.0
Default	n/a	620n	1.0
Response	n/a	625n	1.0
See Also	DWRITE, VARS, WRITE, WRVARS	6270	1.0

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

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

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

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

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

[\ ] ASCII Character Designator

		Product	Rev
Type	Operator (Other)	AT6n00	1.0
Syntax	See below	AT6n50	1.0
Units	n/a	610n	4.0
Range	n/a	615n	1.0
Default	n/a	620n	1.0
Response	n/a	625n	1.0
See Also	VARS, WRITE, WRVARS	6270	1.0

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

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

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

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

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

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

Example:

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

[=] Assignment or Equivalence

		Product	Rev
Type	Operator (Mathematical or Relational)	AT6n00	1.0
Syntax	See below	AT6n50	1.0
Units	n/a	610n	4.0
Range	n/a	615n	1.0
Default	n/a	620n	1.0
Response	n/a	625n	1.0
See Also	[>], [>=], [<], [<=], [<>], [AND], IF, [OR], UNTIL, VAR, VARB, VARS, WAIT, WHILE	6270	1.0

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

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

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

[>] Greater Than

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

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

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

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

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

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

[>=] Greater Than or Equal

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

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

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

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

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

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

[<] Less Than

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

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

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

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

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

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

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

[<=] Less Than or Equal

		Product	Rev
Type	Operator (Relational)		
Syntax	See below	AT6n00	1.0
Units	n/a	AT6n50	1.0
Range	n/a	610n	4.0
Default	n/a	615n	1.0
Response	n/a	620n	1.0
See Also	[=], [>], [<], [>=], [<>], [AND], IF, [OR], UNTIL, WAIT, WHILE	625n	1.0
		6270	1.0

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

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

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

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

[<>] Not Equal

		Product	Rev
Type	Operator (Relational)		
Syntax	See below	AT6n00	1.0
Units	n/a	AT6n50	1.0
Range	n/a	610n	4.0
Default	n/a	615n	1.0
Response	n/a	620n	1.0
See Also	[=], [>=], [<], [<=], [AND], IF, [OR], UNTIL, WAIT, WHILE	625n	1.0
		6270	1.0

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

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

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

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

[( )] Operation Priority Level

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

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

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

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

[+] Addition

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

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

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

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

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

[–] Subtraction

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

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

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

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

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

[*] Multiplication

		Product	Rev
Type	Operator (Mathematical)	AT6n00	1.0
Syntax	See Below	AT6n50	1.0
Units	n/a	610n	4.0
Range	n/a	615n	1.0
Default	n/a	620n	1.0
Response	n/a	625n	1.0
See Also	[=], [()], [+], [-], [/], [SQRT], VAR, VARB	6270	1.0

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

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

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

[/] Division

		Product	Rev
Type	Operator (Mathematical)	AT6n00	1.0
Syntax	See Below	AT6n50	1.0
Units	n/a	610n	4.0
Range	n/a	615n	1.0
Default	n/a	620n	1.0
Response	n/a	625n	1.0
See Also	[=], [()], [+], [-], [*], [SQRT], VAR, VARB	6270	1.0

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

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

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

DIVISION BY ZERO IS NOT ALLOWED.

[&] Boolean And

		Product	Rev
Type	Operator (Bitwise)	AT6n00	1.0
Syntax	See Below	AT6n50	1.0
Units	n/a	610n	4.0
Range	n/a	615n	1.0
Default	n/a	620n	1.0
Response	n/a	625n	1.0
See Also	[=], [|], [~], [^], [<<], [>>], VAR, VARB	6270	1.0

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

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

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

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

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

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

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

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

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

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

[|] Boolean Inclusive Or

		Product	Rev
Type	Operator (Bitwise)	AT6n00	1.0
Syntax	See Below	AT6n50	1.0
Units	n/a	610n	4.0
Range	n/a	615n	1.0
Default	n/a	620n	1.0
Response	n/a	625n	1.0
See Also	[=], [&], [~], [^], [<<], [>>], VAR, VARB	6270	1.0

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

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

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

Example: VAR1=5 | -1
Result: VAR1=1

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

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

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

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

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

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

[^] Boolean Exclusive Or

		Product	Rev
Type	Operator (Bitwise)	AT6n00	1.0
Syntax	See Below	AT6n50	1.0
Units	n/a	610n	4.0
Range	n/a	615n	1.0
Default	n/a	620n	1.0
Response	n/a	625n	1.0
See Also	[=], [&], [~], [	], [<<], [>>], VAR, VARB	6270	1.0

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

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

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

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

Continued on next page:

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

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

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

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

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

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

[~()] Boolean Not

		Product	Rev
Type	Operator (Bitwise)	AT6n00	1.0s
Syntax	See Below	AT6n50	1.0
Units	n/a	610n	4.0
Range	n/a	615n	1.0
Default	n/a	620n	1.0
Response	n/a	625n	1.0
See Also	[=], [&], [^], [|], [<<], [>>], VAR, VARB	6270	1.0

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

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

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

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

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

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

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

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

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

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

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

Type	Operator (Bitwise
Syntax	See Below
Units	n/a
Range	n/a
Default	n/a
Response	n/a
See Also	[=], [&], [^], [|], [~], [>>], VAR, VARB

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

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

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

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

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

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

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

Type	Operator (Bitwise)
Syntax	See Below
Units	n/a
Range	n/a
Default	n/a
Response	n/a
See Also	[=], [&], [^], [|], [~], [<<], VAR, VARB

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

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

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

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

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

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

[Send Response to Both COM Ports

		Product	Rev
Type	Communication Interface	AT6n00	n/a
Syntax	<!> [<command><field1>	AT6n50	n/a
Units	n/a	610n	4.0
Range	n/a	615n	4.1
Default	n/a	620n	4.1
Response	n/a	625n	4.1
See Also	PORT,], ECHO, EOL, EOT, BOT	6270	4.1

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

NOTE: If COM1 and COM2 are not clearly labeled on your product, COM1 is the RS-232 connector (or Rx, Tx, GND terminals on the AUX connector); COM2 is the RP240 connector.

Example

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

] Send Response to Alternate COM Port

		Product	Rev
Type	Communication Interface	AT6n00	n/a
Syntax	<!> [<command><field1>	AT6n50	n/a
Units	n/a	610n	4.0
Range	n/a	615n	4.1
Default	n/a	620n	4.1
Response	n/a	625n	4.1
See Also	PORT, [, ECHO, EOL, EOT, BOT	6270	4.1

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

NOTE: If COM1 and COM2 are not clearly labeled on your product, COM1 is the RS-232 connector (or Rx, Tx, GND terminals on the AUX connector); COM2 is the RP240 connector.

Example

```
PORT1         ; Select COM1
TER           ; Transfer TER Status to port COM1
]TAS          ; Transfer TAS Status to port COM2
```

A — Acceleration

		Product	Rev
Type	Motion	AT6n00	1.0
Syntax	<!><@><a>A<r>,<r>,<r>,<r>	AT6n50	1.0
Units	r = units/sec/sec	610n	4.0
Range	0.00025 - 24,999,999 (depending on the scaling factor)	615n	1.0
Default	10.0000	620n	1.0
Response	A: *A10.0000,10.0000,10.0000,10.0000 1A: *A10.0000	625n	1.0
See Also	[A], AA, AD, ADA, DRES, ERES, GO, MC, SCALE, SCLA, TSTAT	6270	1.0

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

Steppers: The entered value is always in reference to motor steps, not encoder steps, regardless of the ENC command setting. If scaling is not enabled (SCALEØ), the acceleration value is entered in motor revs/sec/sec; this value is internally multiplied by the drive resolution (DRES) value to obtain an acceleration value in motor steps/sec/sec for the motion trajectory calculations. If scaling is enabled (SCALE1), the entered acceleration value is internally multiplied by the acceleration scaling factor (SCLA) to convert user units/sec/sec to motor steps/sec/sec.

Servos: If scaling is not enabled (SCALEØ), the acceleration value is entered in encoder or resolver revs/sec/sec, LDT inches/sec/sec, or ANI volts/sec/sec; encoder or resolver and LDT values are internally multiplied by the encoder resolution (ERES) value or the LDT resolution (LDTRES) value to obtain an acceleration value in steps/sec/sec for the motion trajectory calculations. If scaling is enabled (SCALE1), the entered acceleration value is internally multiplied by the acceleration scaling factor (SCLA) to convert user units/sec/sec to encoder, resolver, LDT, or ANI steps/sec/sec.

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

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

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

Example:

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

[A] Acceleration Assignment

		Product	Rev
Type	Assignment or Comparison	AT6n00	1.0
Syntax	See below	AT6n50	1.0
Units	units/sec/sec	610n	4.0
Range	0.00025 - 24,999,999 (depending on the scaling factor)	615n	1.0
Default	n/a	620n	1.0
Response	n/a	625n	1.0
See Also	A, AA, AD, ADA, DRES, ERES, GO, SCALE, SCLA	6270	1.0

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

Syntax: VARn=aA, where n is the variable number, and a is the axis number, or [A] can be used in an expression such as IF(1A<25000). When assigning the acceleration value to a variable, an axis specifier must always precede the assignment (A) command or it defaults to axis 1 (e.g., VAR1=1A). When making a comparison to the programmed acceleration, an axis specifier must also be used (e.g., IF(1A<20000)). The (A) value used in any comparison, or in any assignment statement is the programmed (A) value.

Steppers: The acceleration value is always in reference to motor steps, not encoder steps, regardless of the ENC command setting. If scaling is not enabled (SCALE0), the acceleration value represents motor revs/sec/sec; this value is internally multiplied by the drive resolution (DRES) value to obtain an acceleration value in motor steps/sec/sec for the motion trajectory calculations. If scaling is enabled (SCALE1), the acceleration value is internally multiplied by the acceleration scaling factor (SCLA) to convert user units/sec/sec to motor steps/sec/sec.

Servos: If scaling is not enabled (SCALE0), the acceleration value is entered in encoder or resolver revs/sec/sec, LDT inches/sec/sec, or ANI volts/sec/sec; encoder or resolver and LDT values are internally multiplied by the encoder resolution (ERES) value or the LDT resolution (LDTRES) value to obtain an acceleration value in steps/sec/sec for the motion trajectory calculations. If scaling is enabled (SCALE1), the entered acceleration value is internally multiplied by the acceleration scaling factor (SCLA) to convert user units/sec/sec to encoder, resolver, LDT, or ANI steps/sec/sec.

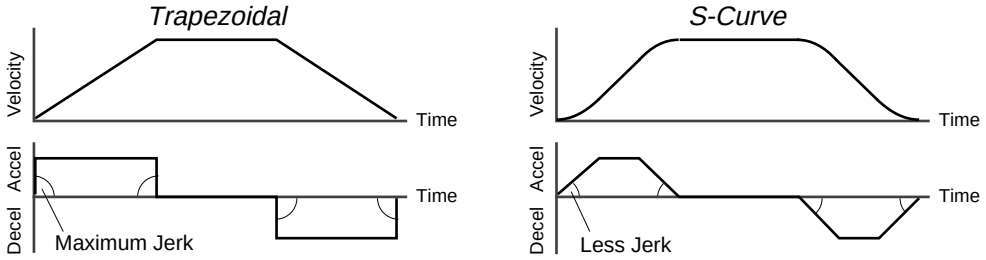
Example:

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

AA — Average Acceleration

		Product	Rev
Type	Motion (S-Curve)	AT6n00	n/a
Syntax	<!><@><a>AA<r>,<r>,<r>,<r>	AT6n50	1.0
Units	r = units/sec/sec	610n	n/a
Range	0.00025 - 24999999 (depending on the scaling factor)	615n	1.0
Default	10.00 (trapezoidal profiling is default, where AA tracks A)	620n	n/a
Response	AA: *AA10.0000,10.0000,10.0000,10.0000 1AA: *1AA10.0000	625n	1.0
See Also	A, AD, ADA, SCALE, SCLA	6270	1.0

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



The values for the maximum accel (A) and average accel (AA) commands determine the characteristics of the S-curve. To smooth the acceleration ramp, you must enter an AA command value that satisfies this equation: 1/2 A ≤ AA < A. The following conditions are possible:

Acceleration Setting	Profiling Condition
AA > 1/2 A, but AA < A	S-curve profile with a variable period of constant acceleration
AA = 1/2 A	*Pure S-curve* (no period of constant acceleration—smoothest motion)
AA = A	Trapezoidal profile (but can be changed to an S-curve by specifying a new AA value less than A)
AA < 1/2 A; or AA > A	When you issue the GO command, the move will not be executed and an error message, *INVALID CONDITIONS FOR S_CURVE ACCELERATION–FIELD *n*, will be displayed.
AA = zero	S-curve profiling is disabled. Trapezoidal profiling is enabled. AA tracks A, & ADA tracks AD. (*Track* means the command's value will match the other command's value and will continue to match whatever the other command's value is set to.)
No AA value ever entered	Profile will default to trapezoidal. AA tracks A.

While programming S-curves, if you never change the maximum or average deceleration (AD or ADA) commands, ADA will track AA. However, once you change AD, ADA will no longer track changes in AA.

NOTE

Once you enter an AA value that is ≠ zero and ≠ A, S-curve profiling is enabled **only for standard moves** (e.g., not for contouring, which requires the PADA and/or PAA commands). All subsequent standard moves for that axis must comply with this equation: 1/2 A ≤ AA < A.

Increasing the AA value above the pure S-curve level (AA > 1/2 A), the time required to reach the target velocity and the target distance is decreased. However, increasing AA also increases jerk.

The calculation for determining S-curve average accel and decel move times is as follows:

(A_{avg} = average accel or decel value):

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

Scaling affects the average acceleration (AA) the same as it does for the maximum acceleration (A).

*** For a more in-depth discussion on S-curve profiling, refer to the Custom Profiling chapter in the *6000 Series Programmer's Guide*.

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

In the example below, axis 1 executes a pure S-curve and takes 1 second to reach a velocity of 5 rps; axis 2 executes a trapezoidal profile and takes 0.5 seconds to reach a velocity of 5 rps.

Example:

```
SCALE0          ; Disable scaling
@MA0            ; Select incremental positioning mode
@D40000         ; Set distances to 40,000 positive-direction steps
A10,10          ; Set max. accel to 10 rev/sec/sec (axes 1 and 2)
AA5,10          ; Set avg. accel to 5 rev/sec/sec on axis 1,
                ; and 10 rev/sec/sec on axis 2
AD10,10         ; Set max. decel to 10 rev/sec/sec (axes 1 and 2)
ADA5,10         ; Set avg. decel to 5 rev/sec/sec on axis 1,
                ; and 10 rev/sec/sec on axis 2
V5,5            ; Set velocity to 5 rps on axes 1 and 2
GO11            ; Execute motion on axes 1 and 2
```

AD — Deceleration

		Product	Rev
Type	Motion	AT6n00	1.0
Syntax	<!><@><a>AD<r>,<r>,<r>,<r>	AT6n50	1.0
Units	r = units/sec/sec	610n	4.0
Range	0.00025 - 24,999,999 (depending on the scaling factor)	615n	1.0
Default	10.0000 (AD tracks A)	620n	1.0
Response	AD: *AD10.0000,10.0000,10.0000,10.0000 1AD: *AD10.0000	625n	1.0
See Also	[A], A, AA, ADA, DRES, ERES, GO, MC, SCALE, SCLA, TSTAT	6270	1.0

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

Steppers: The entered value is always in reference to motor steps, not encoder steps, regardless of the ENC command setting. If scaling is not enabled (SCALEØ), the deceleration value is entered in motor revs/sec/sec; this value is internally multiplied by the drive resolution (DRES) value to obtain an deceleration value in motor steps/sec/sec for the motion trajectory calculations. If scaling is enabled (SCALE1), the entered deceleration value is internally multiplied by the deceleration scaling factor (SCLA) to convert user units/sec/sec to motor steps/sec/sec .

Servos: If scaling is not enabled (SCALEØ), the deceleration value is entered in encoder or resolver revs/sec/sec, LDT inches/sec/sec, or ANI volts/sec/sec; encoder or resolver and LDT values are internally multiplied by the encoder resolution (ERES) value or the LDT resolution (LDTRES) value to obtain an acceleration value in steps/sec/sec for the motion trajectory calculations. If scaling is enabled (SCALE1), the entered deceleration value is internally multiplied by the deceleration scaling factor (SCLA) to convert user units/sec/sec to encoder, resolver, LDT, or ANI steps/sec/sec.

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

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

ON-THE-FLY CHANGES: You can change deceleration *on the fly* (while motion is in progress) in two ways. One way is to send an immediate deceleration command (!AD) followed by an immediate go

command (!GO). The other way is to enable the continuous command execution mode (COMEXC1) and execute a buffered deceleration command (AD) followed by a buffered go command (GO).

Example:

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

[AD] Deceleration Assignment

		Product	Rev
Type	Assignment or Comparison	AT6n00	1.0
Syntax	See below	AT6n50	1.0
Units	units/sec/sec	610n	4.0
Range	0.00025 - 24,999,999 (depending on the scaling factor)	615n	1.0
Default	n/a	620n	1.0
Response	n/a	625n	1.0
See Also	[A], A, AA, AD, ADA, DRES, ERES, GO, SCALE, SCLA	6270	1.0

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

Syntax: VARn=aAD where n is the variable number, and a is the axis number, or [AD] can be used in an expression such as IF(1AD<25000). When assigning the deceleration value to a variable, an axis specifier must always precede the assignment (AD)command or it defaults to axis 1 (e.g., VAR1=1AD). When making a comparison to the programmed deceleration, an axis specifier must also be used (e.g., IF(1AD<20000)). The (AD) value used in any comparison, or in any assignment statement is the programmed (AD) value.

Steppers: The value is always in reference to motor steps, not encoder steps, regardless of the ENC command setting. If scaling is not enabled (SCALE0), the deceleration value represents motor revs/sec/sec; this value is internally multiplied by the drive resolution (DRES) value to obtain an deceleration value in motor steps/sec/sec for the motion trajectory calculations. If scaling is enabled (SCALE1), the deceleration value is internally multiplied by the deceleration scaling factor (SCLA) to convert user units/sec/sec to motor steps/sec/sec.

Servos: If scaling is not enabled (SCALE0), the deceleration value is entered in encoder or resolver revs/sec/sec, LDT inches/sec/sec, or ANI volts/sec/sec; encoder or resolver and LDT values are internally multiplied by the encoder resolution (ERES) value or the LDT resolution (LDTRES) value to obtain an acceleration value in steps/sec/sec for the motion trajectory calculations. If scaling is enabled (SCALE1), the entered deceleration value is internally multiplied by the deceleration scaling factor (SCLA) to convert user units/sec/sec to encoder, resolver, LDT, or ANI steps/sec/sec.

Example:

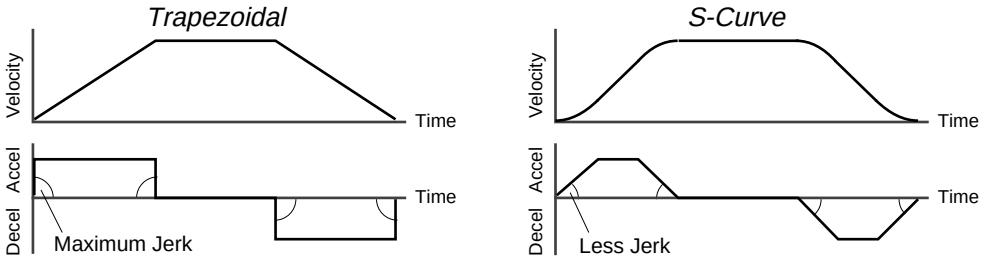
```
IF(2AD<25000)   ; If the deceleration on axis 2 is less than 25000 units/sec/sec,
                ; then do the statements between the IF and NIF
VAR1=2AD*2      ; Variable 1 = deceleration of axis 2 times 2
AD,(VAR1)       ; Set the deceleration on axis 2 to the value of variable 1
NIF         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

ADA — Average Deceleration

Type	Motion (S-Curve)
Syntax	<!><@><a>ADA<r>,<r>,<r>,<r>
Units	r = units/sec/sec
Range	0.00025 - 24999999 (depending on the scaling factor)
Default	10.00 (ADA tracks AA)
Response	ADA: *ADA10.0000,10.0000,10.0000,10.0000 1ADA: *1ADA10.0000
See Also	A, AA, AD, SCALE, SCLA

Product	Rev
AT6n00	n/a
AT6n50	1.0
610n	n/a
615n	1.0
620n	n/a
625n	1.0
6270	1.0

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



The values for the maximum decel (AD) and average decel (ADA) commands determine the characteristics of the S-curve. To smooth the deceleration ramp, you must enter an ADA command value that satisfies this equation: 1/2 AD ≤ ADA < AD. The following conditions are possible:

Deceleration Setting	Profiling Condition
ADA > 1/2 AD, but ADA < AD	S-curve profile with a variable period of constant deceleration
ADA = 1/2 AD	*Pure S-curve* (no period of constant deceleration—smoothest motion)
ADA = AD	Trapezoidal profile (but can be changed to S-curve by specifying a new ADA value less than AD)
ADA < 1/2 AD; or ADA > AD	When you issue the GO command, the move will not be executed and an error message, *INVALID CONDITIONS FOR S_CURVE ACCELERATION–FIELD *n*, will be displayed.
ADA = zero	Upon entering the ADAØ command, an error message, *INVALID DATA–FIELD *n*, will be displayed.
AD = zero	AD tracks A and ADA tracks AA, whether or not the acceleration is an s-curve.
S-curve profiling with AA, and no ADA or AD ever entered	ADA will always match the AA command value (identical S-curve accel and decel profiles). When you change AD, ADA will no longer match changes in AA.

NOTE

Once you enter an ADA value that is ≠ zero or ≠ AD, S-curve profiling is enabled **only for standard move decelerations** (e.g., not for contouring decelerations, which require the PADA command). All subsequent standard moves for that axis must comply with this equation: 1/2 AD ≤ ADA < AD.

Increasing the ADA value above the pure S-curve level (ADA > 1/2 AD), the time required to reach the target velocity and the target distance is decreased. However, increasing ADA also increases jerk.

The calculation for determining S-curve average accel and decel move times is as follows:

(A_{avg} = average accel or decel value):

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

Scaling affects the average deceleration (ADA) the same as it does for the maximum deceleration (AD).

*** For a more in-depth discussion on S-curve profiling, refer to the Custom Profiling chapter in the *6000 Series Programmer's Guide*.

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

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

Example:

```
SCALE0          ; Disable scaling
@MA0            ; Select incremental positioning mode
@D40000         ; Set distances to 40,000 positive-direction steps
A10,10          ; Set max. accel to 10 rev/sec/sec (axes 1 and 2)
AA5,10          ; Set avg. accel to 5 rev/sec/sec on axis 1,
                ; and 10 rev/sec/sec on axis 2
AD10,10         ; Set max. decel to 10 rev/sec/sec (axes 1 and 2)
ADA5,10         ; Set avg. decel to 5 rev/sec/sec on axis 1,
                ; and 10 rev/sec/sec on axis 2
V5,5            ; Set velocity to 5 rps on axes 1 and 2
GO11            ; Execute motion on axes 1 and 2
```

ADDR — Multiple Unit Auto-Address

		Product	Rev
Type	Controller Configuration	AT6n00	n/a
Syntax	<!>ADDR<i>	AT6n50	n/a
Units	i = axis number	610n	4.0
Range	0 to 99	615n	1.0
Default	Defaults to the DIP switch setting (default DIP switch setting is 0)	620n	1.5
Response	ADDR: *ADDR0	625n	1.0
See Also	E, PORT	6270	1.0

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

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

Setting `ADDR` to `Ø` re-enables the unit's address configured on its internal DIP switch.

RS-232C Daisy Chain:

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

RS-485 multi-drop:

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

1. Connect the unit that is to be unit #1 and transmit the `Ø_ADDR1` command to it.
2. Connect the unit that is to be unit #2 and transmit the `Ø_ADDR2` command to it.
3. Connect the unit that is to be unit #3 and transmit the `Ø_ADDR3` command to it.
4. Connect the unit that is to be unit #4 and transmit the `Ø_ADDR4` command to it.

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

To send a 6000 command from the master unit to a specific unit in the multi-drop, prefix the command with the unit address and an underscore (e.g., `3_OUTØ` turns off output #1 on unit #3). The master unit (if it is not a 6000 product) may receive data from a multi-drop unit.

For more information on controlling multiple 6000 Series controllers in an RS-232 daisy-chain or RS-485 multi-drop, refer to the *6000 Series Programmer's Guide*.

Example:

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

[AND] And

		Product	Rev
Type	Operator (logical)	AT6n00	1.0
Syntax	See below	AT6n50	1.0
Units	n/a	610n	4.0
Range	n/a	615n	1.0
Default	n/a	620n	1.0
Response	n/a	625n	1.0
See Also	IF, [NOT], [OR], REPEAT, UNTIL, WAIT, WHILE	6270	1.0

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

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

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

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

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

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

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

[ANI] Analog Input Value (-ANI Option Only)

		Product	**Rev**
Type	Assignment or comparison	AT6n00	n/a
Syntax	See below	AT6n50-ANI	1.0
Units	n/a	610n	n/a
Range	n/a	615n-ANI	1.0
Default	n/a	620n	n/a
Response	n/a	625n-ANI	1.1
See Also	[CA], [FB], [PANI], [PCA], SFB, TANI, TANV, TFB, TPANI, TPCA	6270-ANI	1.0

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

The ANI value is derived from the voltage applied to the corresponding analog input and ground. The analog value is determined from a 14-bit analog-to-digital converter. The minimum voltage response is -10.000VDC, the maximum voltage response is +10.000VDC.

Syntax: VARn=aANI where n is the variable number, and a is analog input number 1-4, or ANI can be used in an expression such as IF(1ANI=2.3). An analog input number specifier must precede the ANI command, or else it will default to input 1 (e.g., 1ANI, 2ANI, etc.).

Example:

```
VAR2=2ANI       ; Voltage value at 6250-ANI's analog input 2 is assigned to
                ; variable 2
IF(1ANI<8.2)    ; If voltage value at 6250-ANI's analog input 1 < 8.2V, do the
                ; commands between the IF statement and the NIF statement.
TREV            ; Transfer revision level
NIF             ; End if statement
```

ANIPOL — ANI Input Polarity *(-ANI Option Only)*

Type	ANI; Controller Configuration
Syntax	<!><@><a>ANIPOL<b><b>
Units	b = polarity bit
Range	0 (normal polarity), 1 (reverse polarity) or X (don't change)
Default	0
Response	ANIPOL *ANIPOL00 1AINPOL *1ANIPOL0
See Also	CMDDIR, [FB], FOLMAS, [PANI], [PCA], [PER], PSET, SFB, TFB, TPANI, TPCA, TPER

Product	Rev
AT6n00	n/a
AT6n50-ANI	n/a
610n	n/a
615n	n/a
620n	n/a
625n-ANI	n/a
6270-ANI	3.0

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

If the ANI input is counting in the wrong direction, you may reverse the polarity with the ANIPOL command (see programming example below). This allows you to reverse the counting direction without having to change the actual mounting of the ANI input. For example, if the ANI on axis 2 counted in the wrong direction, you could issue the ANIPOLx1 command to correct the polarity.

Immediately after issuing the ANIPOL command, the sign of the ANI counts or voltage values (including all ANI position/voltage registers) is reversed. The polarity is immediately changed whether or not ANI feedback is currently selected with the SFB command. The polarity reversal affects the values of TPANI, PANI, TPCA, PCA, TFB, FB, TPER, and PER; it does not affect the values of TANI, ANI, TCA, or CA.

NOTE

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

The ANIPOL command is automatically saved in non-volatile RAM (stand-alone products only).

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

Example:

```
SFB1,2          ; Select encoder feedback for axis 1 and ANI feedback for axis 2
2TPANI          ; *2TPANI+1.254 (response indicates ANI #2 at position +1.254)
ANIPOLx1        ; Reverse ANI polarity on axis 2
2TPANI          ; *2TPANI-1.254 (response indicates ANI #2 at position -1.254)
```

[ANV] — Analog Input Value

Type	Assignment or Comparison
Syntax	See below
Units	n/a
Range	n/a
Default	n/a
Response	n/a
See Also	ANVO, ANVOEN, JOY, TANV, TINO, VAR

Product	Rev
AT6n00	1.0
OEM-AT6n00	n/a
AT6n50	1.0
610n	n/a
615n	n/a
620n	1.0
625n	1.0
6270	1.0

The Analog Input Value (ANV) command is used to assign an analog input value to a variable, or to make a comparison against another value. When using ANV, an analog input channel specifier must always precede

the ANV command or else it will default to channel 1. The analog channel specifier can be 1, 2, 3, or 4 (1ANV, 2ANV, 3ANV, or 4ANV), for analog input channels 1, 2, 3, and 4, respectively. *The number of analog input channels available varies by product.*

Syntax: VARn=aANV where n is the variable number, and a is the analog channel, or [ANV] can be used in an expression such as IF(1ANV=2.3).

The ANV command will provide a voltage value from the analog channel queried. The value is derived from the voltage between the corresponding analog channel and ground. The minimum voltage response will be Ø VDC, while the maximum voltage response will be 2.5 VDC. Joystick connector pin outs are provided below.

Pin # on Joystick Connector	Function	Pin # on Joystick Connector	Function
1	Analog Channel 1	15	Axes Select
2	Analog Channel 2	16	Velocity Select
3	Analog Channel 3	17	Joystick Release
4	Analog Channel 4 *(product dependent)*	18	Joystick Trigger
8	Shield	19	Joystick Auxiliary
14	Ground	23	+5VDC (out)

Example:

```
VAR2=4ANV       ; Voltage value for analog channel 4 is assigned to variable 2
IF(1ANV<2.4)    ; If voltage value for analog channel 1 is less than 2.4 volts,
                ; do the commands between the IF statement and the NIF statement
TREV            ; Transfer revision level
NIF             ; End IF statement
```

ANVO — Analog Input Voltage Override

Type	Input or Joystick or Program Debug Tool
Syntax	<!><@><a>ANVO<r>,<r>,<r>,<r>
Units	r = volts for analog channels 1, 2, 3, & 4, respectively
Range	0 - 2.500
Default	1.244
Response	ANVO: *ANVO1.244,1.244,1.244,1.244 1ANVO: *ANVO1.244
See Also	[ANV], ANVOEN, TANV

Product	Rev
AT6n00	2.1
OEM-AT6n00	n/a
AT6n50	1.0
610n	n/a
615n	n/a
620n	2.1
625n	1.1
6270	1.0

After enabling the Analog Input Voltage Override function with the ANVOEN1 command, you can use the Analog Input Voltage Override (ANVO) command to override the existing voltage on the analog input channels (on the **JOYSTICK** connector). The ANVO values are used in any command or function that references the analog input channel, but only those channels for which the override function has been enabled with the ANVOEN command (see example below).

Overriding the analog input channels allows you to simulate input values for program debugging purposes. Another use for the ANVO command may be to use it in an ERRORP program to override the analog input voltage in response to a fault.

Example:

```
ANVO.96,1.85,1.05,2.35  ; Set analog input override values to 0.96V, 1.85V,
                        ; 1.05V & 2.35V for analog input channels 1-4
ANVOEN1001              ; Enable analog input voltage override on
                        ; channels 1 and 4 only
TANV                    ; Transfer the values of the analog input channels.
                        ; Response should be: *TANV.96,1.244,1.244,2.35
                        ; (Note that only channels 1 and 4 reflect the values
                        ; specified with the ANVO command. Channels 2 and 3 are
                        ; not overridden.)
```

ANVOEN — Analog Input Voltage Override Enable

Type	Input or Joystick or Program Debug Tool
Syntax	<!><@><a>ANVOEN<b><b><b><b>
Units	n/a
Range	b = 0 (disable), 1 (enable) or X (don't change)
Default	0
Response	ANVOEN: *ANVOEN0000 1ANVOEN: *ANVOEN0
See Also	[ANV], ANVO, TANV

Product	Rev
AT6n00	2.1
OEM-AT6n00	n/a
AT6n50	1.0
610n	n/a
615n	n/a
620n	2.1
625n	1.1
6270	1.0

The Analog Input Voltage Override Enable (ANVOEN) command determines whether the analog input voltages are the actual hardware values, or are overridden by the ANVO values. If the ANVOEN value is Ø, then the actual hardware value is used for that analog input channel. If the ANVOEN value is 1, then the ANVO value is used. *The number of analog input channels available varies by product.*

The joystick release input (pin #17 on the **JOYSTICK** connector) is not monitored when ANVOEN is enabled for any analog input channel. Thus, you can enter the joystick mode and simulate joystick operations.

Example: (see ANVO example)

[AS] — Axis Status

Type	Assignment or Comparison
Syntax	See below
Units	n/a
Range	n/a
Default	n/a
Response	n/a
See Also	[ASX], GOWHEN, INDUST, LDTUPD, SMPER, TAS, TASF, TRGFN, TSTAT, VARB

Product	Rev
AT6n00	1.4
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.5
625n	1.0
6270	1.0

The Axis Status (AS) command is used to assign the axis status bits for a specific axis to a binary variable, or to make a comparison against a binary or hexadecimal value.

To make a comparison against a binary value, the letter b (b or B) must be placed in front of the value that the axis status is being compared against. The binary value itself must only contain ones, zeros, or Xs (1, Ø, X, x). To make a comparison against a hexadecimal value, the letter h (h or H) must be placed in front of the value that the axis status is being compared against. The hexadecimal value itself must only contain the letters A through F, and the numbers Ø through 9. When using AS, an axis specifier must always proceed it, or else it will default to axis 1. Valid axis specifiers are 1, 2, 3, or 4 (1AS, 2AS, 3AS, or 4AS). The function of each axis status bit is shown below. An "x" identifies products to which the function is applicable.

Bit # (left to right)	Function (1/Ø)	AT6n00	OEM-AT6n00	AT6250	AT6450	610n	615n	620n	625n	6270
1	Moving/Not Moving	x	x	x	x	x	x	x	x	x
2	Negative/positive-direction	x	x	x	x	x	x	x	x	x
3	Accelerating/Not Accelerating	x	x	x	x	x	x	x	x	x
4	At Velocity/Not at Velocity	x	x	x	x	x	x	x	x	x
5	Home Successful (HOM) YES/NO	x	x	x	x	x	x	x	x	x
6	Absolute/Incremental (MA)	x	x	x	x	x	x	x	x	x
7	Continuous/Preset (MC)	x	x	x	x	x	x	x	x	x
8	Jog Mode/Not Jog Mode (JOG)	x	x	x	x	n/a	n/a	x	x	x

Bit # (left to right)	Function (1/Ø)	AT6n00	OEM-AT6n00	AT6250	AT6450	610n	615n	620n	625n	6270
9	Joystick Mode/Not Joystick Mode (JOY)	x	n/a	x	x	n/a	n/a	x	x	x
10	Encoder Step Mode/Motor Step Mode (ENC)	x	n/a	n/a	n/a	x	n/a	x	n/a	n/a
11	Position Maintenance (EPM) ON/OFF	x	n/a	n/a	n/a	x	n/a	x	n/a	n/a
12	Stall Detected (ESTALL) YES/NO	x	n/a	n/a	n/a	x	n/a	x	n/a	n/a
13	Drive Shut Down occurred YES/NO	x	n/a	x	x	x	x	x	x	x
14 *	Drive Fault occurred YES/NO	x	n/a	x	x	x	x	x	x	x
15	Positive-direction Hardware Limit Hit YES/NO	x	x	x	x	x	x	x	x	x
16	Negative-direction Hardware Limit Hit YES/NO	x	x	x	x	x	x	x	x	x
17	Positive-direction Software Limit Hit YES/NO	x	x	x	x	x	x	x	x	x
18	Negative-direction Software Limit Hit YES/NO	x	x	x	x	x	x	x	x	x
19	Within Deadband (EPMDB) YES/NO	x	n/a	n/a	n/a	x	n/a	x	n/a	n/a
20	In Position (COMEXP) YES/NO	x	n/a	n/a	n/a	n/a	n/a	x	n/a	n/a
21	Distance Streaming Mode (STREAM1) YES/NO	x	x	n/a	n/a	n/a	n/a	n/a	n/a	n/a
22	Velocity Streaming Mode (STREAM2) YES/NO	x	x	n/a	n/a	n/a	n/a	n/a	n/a	n/a
23	Position Error Exceeded (SMPER) YES/NO	n/a	n/a	x	x	n/a	x	n/a	x	x
24 **	In Target Zone (STRGTD & STRGTV) YES/NO	x	n/a	x	x	x	x	x	x	x
25	Target Zone Timeout occurred (STRGTT) YES/NO	x	n/a	x	x	x	x	x	x	x
26 ***	Motion suspended pending GOWHEN YES/NO	x	x	x	x	x	x	x	x	x
27	LDT Position Read Error YES/NO	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	x
28 ****	Registration move initiated by trigger since last GO command	x	x	x	x	x	x	x	x	x
29	RESERVED	---	---	---	---	---	---	---	---	---
30	Pre-emptive (OTF) GO or Registration profile not possible	x	x	x	x	x	x	x	x	x
31	RESERVED	---	---	---	---	---	---	---	---	---
32	RESERVED	---	---	---	---	---	---	---	---	---

* The input functions must be enabled (INFEN1) before a drive fault will be recognized.
610n only: ASX bits 1-3 provide specific causes for the fault.

** This bit is set only after the successful completion of a move.

*** This bit is cleared if GOWHEN condition is true, or if STOP (!S) or KILL (!K or ^K) is executed.

**** This bit is cleared with the next GO command.

Syntax: VARBn=aAS where n is the binary variable number and a is the axis identifier, or [AS] can be used in an expression such as IF(1AS=b11Ø1), or IF(1AS=h7F). If it is desired to assign only one bit of the axis status value to a binary variable, instead of all 32, the bit select (.) operator can be used. The bit select, in conjunction with the bit number, is used to specify a specific axis status bit (e.g., VARB1=1AS.12 assigns axis 1 status bit 12 to binary variable 1).

Example:

```
VARB1=1AS           ; Axis status for axis 1 assigned to binary variable 1
VARB2=1AS.12        ; Axis 1 status bit 12 assigned to binary variable 2
VARB2               ; Response, if bit 12 is set to 1, is
                    ; "*VARB2=XXXX_XXXX_XXX1_XXXX_XXXX_XXXX_XXXX_XXXX"
IF(4AS=b111011X11)  ; If the axis status for axis 4 contains 1's for
                    ; inputs 1,2,3,5,6,8,and 9, and a 0 for bit location 4,
                    ; do the IF statement
TREV                ; Transfer revision level
NIF                 ; End if statement
IF(2AS=h7F00)       ; If the axis status for axis 2 contains 1's for
                    ; inputs 1,2,3,5,6,7,and 8, and 0's for every other bit
                    ; location, do the IF statement
TREV                ; Transfer revision level
NIF                 ; End if statement
```

[ASX] Extended Axis Status

Type	Assignment or Comparison
Syntax	See below
Units	n/a
Range	n/a
Default	n/a
Response	n/a
See Also	TASX, TASXF, [AS], TAS, TASF, VARB

Product	Rev
AT6n00	4.1
AT6n50	4.1
610n	4.0
615n	4.1
620n	4.1
625n	4.1
6270	4.1

The Extended Axis Status (ASX) command is used to assign the axis status bits for a specific axis to a binary variable, or to make a comparison against a binary or hexadecimal value.

To make a comparison against a binary value, the letter b (b or B) must be placed in front of the value that the axis status is being compared against. The binary value itself must only contain ones, zeros, or Xs (1, Ø, X, x). To make a comparison against a hexadecimal value, the letter h (h or H) must be placed in front of the value that the axis status is being compared against. The hexadecimal value itself must only contain the letters A through F, and the numbers Ø through 9. An "x" identifies products to which the function is applicable.

Bit Assignment (left to right)	Function (1 = yes, Ø = no)	AT6n00	OEM-AT6n00	AT6n50	610n	615n	620n	625n	6270
1	Motor Fault (610n only)	---	---	---	x	---	---	---	---
2	Low Voltage (610n only)	---	---	---	x	---	---	---	---
3	Over Temperature Fault (610n only)	---	---	---	x	---	---	---	---
4	Drive Fault Input Active *	x	x	x	x	x	x	x	x
5-32	RESERVED	---	---	---	---	---	---	---	---

* Bit #4 indicates the current hardware state of the drive fault input, whether or not the drive is enabled.

Syntax: VARBn=ASX where n is the binary variable number, or [ASX] can be used in an expression such as IF(ASX=b11ØØ), or IF(ASX=h7Ø). If it is desired to assign only one bit of the axis status value to a binary variable, instead of all 32, the bit select (.) operator can be used. The bit select, in conjunction with the bit number, is used to specify a specific axis status bit (e.g., VARB1=ASX.3 assigns axis 1 status bit 3 to binary variable 1).

Example:

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

[ATAN()] Arc Tangent

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

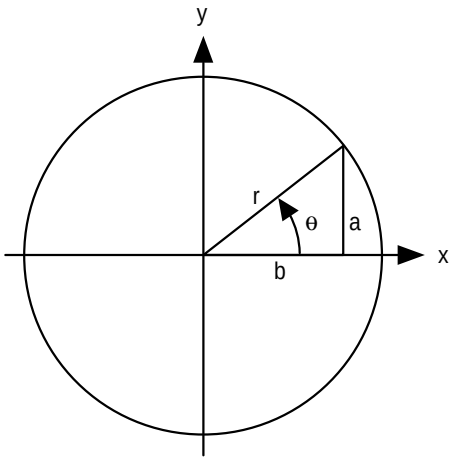
Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

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

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

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

To convert radians to degrees, use the formula: 360° = 2π radians.



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

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

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

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

Example:

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

[b] Binary Identifier

		Product	Rev
Type	Operator (Other)		
Syntax	See below	AT6n00	1.0
Units	n/a	AT6n50	1.0
Range	n/a	610n	4.0
Default	n/a	615n	1.0
Response	n/a	620n	1.0
See Also	[AS], [ER], [h], [IN], [INO], [LIM], [MOV], [OUT], [SS], [US], VARB	625n	1.0
		6270	1.0

This identifier allows you to specify binary values (bit patterns). The letter b must precede the binary value. All other bits not specified are set to zero.

Example:
```
WAIT(IN=b1101)  ; Wait for input pattern
```

BOT Beginning of Transmission Characters

		Product	Rev
Type	Communication Interface		
Syntax	<!>BOT<i>,<i>,<i>	AT6n00	4.1
Units	n/a	AT6n50	4.1
Range	i = 0 - 256	610n	4.0
Default	0,0,0	615n	4.1
Response	BOT: *BOT0,0,0	620n	4.1
See Also	EOT, ERROK, ERRBAD, PORT, DRPCHK,EOL,], [	625n	4.1
		6270	4.1

The Beginning of Transmission Characters (BOT) command designates the characters to be placed at the beginning of every response. Up to 3 characters can be placed before the first line of a multi-line response, or before all single-line responses. The characters are designated with their ASCII equivalent. For example, a carriage return is ASCII 13, a line feed is ASCII 10, a Ctrl-Z is ASCII 26, and no terminating character is designated with a zero. Note that ASCII 256 means ØØ is transmitted.

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

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

BP Set a Program Break Point

		Product	Rev
Type	Program Flow Control or Program Debug Tool		
Syntax	<!>BP<i>	AT6n00	1.0
Units	i = break point number	AT6n50	1.0
Range	1 - 32	610n	4.0
Default	n/a	615n	1.0
Response	n/a	620n	1.0
See Also	BREAK, C, HALT, K, S, [SS], TSS	625n	1.0
		6270	1.0

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

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

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

Example:

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

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

BREAK — Terminate Program Execution

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

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

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

To terminate all program processing, use the HALT command.

Example:

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

C — Continue Command Execution

Type	Program Flow Control
Syntax	!C
Units	n/a
Range	n/a
Default	n/a
Response	n/a
See Also	BP, COMEXR, COMEXS, INFNC, PS, S

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

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

Example:

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

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

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

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

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

When the 6000 Series product processes the !S command, motion on all axes will be stopped. If the desired distance has not been reached, motion can be resumed by issuing the !C command. If motion and command processing are to stop, a Kill (!K) command can be issued.

[CA] Captured ANI Input Voltage

Type	Assignment or Comparison
Syntax	See below
Units	n/a
Range	-10V to +10V
Default	n/a
Response	n/a
See Also	[ANI], INFNC, [PCA], SFB, [SS], SSFR, TANI, TCA, TPCA, TSS

Product	Rev
AT6n00	n/a
AT6n50-ANI	3.3
610n	n/a
615n-ANI	3.3
620n	n/a
625n-ANI	3.3
6270-ANI	3.0

Use the CA command to assign one of the captured ANI analog input register values (captured when trigger A, B, C or D is activated) to a variable, or to make a comparison against another value. Once the captured ANI register value is assigned to a variable, or a comparison is made, the respective voltage capture status (reported with bits 25 - 28 in the TSS or SS commands) is cleared, but the voltage information remains in the register until it is overwritten by a subsequent voltage capture from the trigger input.

The CA value is referenced in volts and does not reflect the affects of scaling (SCLD), position offset (PSET), ANI feedback polarity (ANIPOL), or commanded direction polarity (CMDDIR). To assign/compare the ANI input value as affected by SCLD, PSET, ANIPOL, or CMDDIR, use the PCA command.

Syntax: VARn=aCAc where n is the variable number, a is the axis, and c designates trigger A, B, C or D; or the CA command can be used in an expression such as IF(1CAB>5).

The CA command must be used with an analog input specifier or it will default to analog input 1 (e.g., 1CAA, 2CAB, etc.).

The ANI input value can be captured only by a trigger input signal (trigger A, B, C or D). The appropriate trigger inputs must be defined as *trigger interrupt* inputs with the INFNCi-H command, where i is the number of trigger input (A-D), usually represented by numbers 25-28, respectively. Once defined, an active signal on the specified trigger input will capture the ANI values on all axes. The ANI *voltage* information is stored in registers and is available at the next system update through the use of the CA and TCA commands.

CAPTURE ACCURACY

If ANI feedback is selected with the SFB command, the captured ANI value is interpolated from the last sampled ANI input value and rate of change of the ANI input value, and the time elapsed since the last sample. *The sample rate is determined by the SSFR and INDAX commands (system update rate). The accuracy of the capture is ±50µs x velocity.*

If ANI feedback is NOT selected with the SFB command, the last sampled ANI value is simply stored as the captured ANI value. *The accuracy is one system update period (determined by SSFR and INDAX).*

Example:

```
INFNC26-H       ; Assign trigger input B (TRG-B) as a trigger interrupt input
INFNC25-H       ; Assign trigger input A (TRG-A) as a trigger interrupt input
INFEN1          ; Enable input functions for trigger interrupts
IF(SS.25=b1)    ; If trigger A (input #25) becomes active, execute the commands
                ; between IF and NIF
 VAR1=1CAA      ; Assign captured ANI value on analog input 1 (captured when the
                ; TRG-A input became active) to variable 1
 IF(2CAB<40)    ; If the captured ANI value on analog input 2 (captured when the
                ; TRG-B input became active) is less than 40, do the IF statement
  VAR2=1CAA+10  ; Add 10 to the captured ANI value on analog input 1 (captured
                ; when the TRG-A input became active) and assign the sum to
                ; variable #2
NIF             ; End IF statement
```

CMDDIR Commanded Direction Polarity

		Product	Rev
Type	Controller Configuration	AT6n00	4.2
Syntax	<@><a>CMDDIR<b><b><b><b>	AT6n50	n/a
Units	b = polarity bit	610n	4.0
Range	0 (normal polarity), 1 (reverse polarity) or X (don't change)	615n	3.0
Default	0	620n	4.2
Response	CMDDIR *CMDDIR0000 1CMDDIR *1CMDDIR0	625n	n/a
See Also	ANIPOL, [AS], DRIVE, ENCPOL, [FB], [LDT], LDTPOL, [PANI], [PCE], [PE], [PER], PSET, SFB, TAS, TFB, TLDT, TPANI, TPCE, TPE, TPER	6270	3.0

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

- Before changing the commanded direction polarity, make sure there is a direct correlation between the commanded direction and the direction of the feedback source counts (i.e., a positive commanded direction from the controller must result in positive counts from the feedback device). Refer to the ANIPOL, ENCPOL, or LDTPOL command descriptions for further information.
- 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 (stand-alone products only).

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

[CNT] Counter Value

		Product	Rev
Type	Assignment or Comparison	AT6n00	1.0
Syntax	See below	OEM-AT6n00	n/a
Units	n/a	AT6n50	n/a
Range	n/a	610n	4.0
Default	n/a	615n	n/a
Response	n/a	620n	1.0
See Also	CNTE, CNTINT, CNTR, TCNT	625n	n/a
		6270	n/a

The Counter Value (CNT) command is used to assign a hardware counter value to a variable, or to make a comparison against another value.

Syntax: VARn=aCNT where n is the variable number, and a is the axis number, or [CNT] can be used in an expression such as IF(1CNT<13000)

All encoder inputs can be converted to hardware counters through the use of the CNTE command. Each hardware counter can count up or count down. The direction of count is specified by the signal on the encoder channel B+ and B- connections. A positive differential signal, when measured between B+ and B-, will infer a negative count direction. A negative differential signal, when measured between B+ and B-, will infer a positive count direction. The count itself is determined from the signal on A+ and A-. Each count is registered on the positive edge of a transition for a signal measured between A+ and A-. To reset the counter, apply a positive differential signal to Z+ and Z-, or issue the CNTR command.

If an encoder input has not been defined as a counter input, the CNT command will provide a counter value of zero.

Example:
```
CNTE1000        ; Define encoder input 1 as a counter
CNTR1000        ; Reset encoder input 1
T5              ; Wait 5 seconds
VAR1=1CNT       ; Variable 1 equals the count from encoder channel 1
IF(1CNT<15)     ; If the count is less than 15, do the commands between IF & NIF
 WRVAR1         ; Write out variable 1
NIF             ; End IF statement
```

CNTE — Hardware Up/Down Counter Input

Type	Counter
Syntax	<!><@><a>CNTE<b><b><b><b>
Units	n/a
Range	0 = Encoder, 1 = Counter
Default	0
Response	CNTE: *CNTE0000 1CNTE: *1CNTE0
See Also	[CNT], CNTINT, CNTR, FOLSND, TCNT

Product	Rev
AT6n00	1.0
OEM-AT6n00	n/a
AT6n50	n/a
610n	4.0
615n	n/a
620n	1.0
625n	n/a
6270	n/a

The CNTE command is used to specify if the encoder input is to be used as a counter input. Each encoder input can be used as a counter. The hardware counter can either count up or down. The direction of the count is specified by the signal on the encoder channel B+ and B- connections. A positive differential signal, when measured between B+ and B-, will infer a negative count direction. A negative differential signal, when measured between B+ and B-, will infer a positive count direction. The count itself is determined from the signal on A+ and A-. Each count is registered on the positive edge of a transition for a signal measured between A+ and A-. To reset the counter, apply a positive differential signal to Z+ and Z-, or issue the CNTR command.

If you are going to use the encoder ports for an encoder instead of using it as a counter input, specify CNTE0000 (the default state).

The value of the counter can be accessed at any time through the hardware registers (bus-based products only) or by doing a software transfer (TCNT). The hardware registers in the fast status area provide information on encoder position, but when an encoder input is defined as a counter, the information in the register is a count value.

NOTE: The CNTE1 feature may not be used while the Following Step & Direction feature is enabled (FOLSND1).

Example:
```
CNTE0100        ; Specify the axis 2 encoder port as a hardware counter
```

CNTINT — Counter Value to Interrupt PC-AT

Type	Counter
Syntax	<!>CNTINT<i,i,i>
Units	n/a
Range	First i = 1 to 4, 2nd & 3rd i = -999,999,999 to 999,999,999
Default	0,+0,+0
Response	CNTINT: *CNTINT0,+0,+0
See Also	[CNT], CNTE, CNTR, INTHW, TCNT

Product	Rev
AT6n00	1.0
OEM-AT6n00	n/a
AT6n50	n/a
610n	n/a
615n	n/a
620n	n/a
625n	n/a
6270	n/a

This command sets the high and low values upon which the 6000 controller will interrupt the PC-AT. Only one of the possible four hardware counters can be defined to interrupt the PC-AT. If multiple CNTINT commands are entered, only the last one entered is used.

CNTINT syntax: first <i> determines which encoder (counter) channel to use; second <i> determines the low value to interrupt the PC-AT; third <i> determines the high value to interrupt the PC-AT.

Example:
```
CNTINT4,0,25000  ; If the count for encoder channel 4 ever exceeds 25000, or
                 ; falls below 0, interrupt the PC-AT (count < 0 or count > 25000)
```

CNTR — Reset Hardware Up/Down Counter

Type	Counter
Syntax	<!><@>CNTR<b><b><b><b>
Units	b = 0, 1 or X
Range	0 = don't reset, 1 = reset, X = don't change
Default	n/a
Response	CNTR: No response, all counters will be reset
See Also	[CNT], CNTE, CNTINT, TCNT

Product	Rev
AT6n00	1.0
OEM-AT6n00	n/a
AT6n50	n/a
610n	4.0
615n	n/a
620n	1.0
625n	n/a
6270	n/a

The Reset Hardware Up/Down Counter (CNTR) command is used to clear the value of any encoder registers that were specified as hardware counters with the CNTE command.

The hardware counter can either count up or down. The direction of the count is specified by the signal on the encoder channel B+ and B- connections. The count itself is determined from the signal on A+ and A-. To reset the counter, apply a positive differential signal to Z+ and Z-, or issue the CNTR command.

To specify if an encoder input is to be used as a hardware counter, refer to the CNTE command.

Example:

```
CNTE1011        ; Configure encoder inputs 1, 3, and 4 as hardware counters
CNTR            ; Reset all the hardware counters
CNTR0001        ; Reset hardware counter 4
```

COMEXC — Continuous Command Processing Mode

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

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

This command enables (COMEXC1) or disables (COMEXCØ) Continuous Command Execution Mode. Normally, when a motion command is received, command processing is temporarily paused until the motion is complete. In continuous command execution mode, however, command processing continues while motion is taking place. *Command processing will be slower and **some** motion parameters cannot be changed while motion is in progress. For a complete list of motion parameters that cannot be changed while motion is in progress, refer to the* Restricted Commands During Motion *section in Chapter 1 of the* 6000 Series Programmer's Guide.

This mode is useful in the following situations:

- When trying to check the status of inputs while the 6000 Series product is commanding motion.
- Performing calculations ahead of time, possibly decreasing cycle time.
- Executing buffered on-the-fly acceleration (A, AA), and deceleration (AD, ADA), distance (D), positioning mode (MA & MC), Following ratio (FOLRD & FOLRN), and velocity (V) changes. (The buffered A, AA, AD, ADA, D, FOLRD, FOLRN, MA, MC, or V change can be executed only with a buffered Go (GO) command.) For more information about on-the-fly motion changes, refer to the *Programmer's Guide.*
- Pre-processing the next move while the current move is in progress (see CAUTION note below). This reduces the processing time for the subsequent move to only a few microseconds.

CAUTION: Avoid Executing Moves Prematurely

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

Example:

```
VAR1=2000       ; Set variable 1 = 2000
VAR2=0          ; Set variable 2 = 0
COMEXC1         ; Enable continuous command execution mode
L50             ; Loop 50 times
D50000,(VAR1)   ; Set distance to 50000 units for axis 1, VAR1 value for axis 2
GO1100          ; Initiate motion on axes 1 and 2
; Normally at this point, the 6000 Series Product would wait for the motion on
; axes 1 and 2 to complete before processing the next command. However, with
; continuous command mode enabled, processing will continue with the statements
; that follow.
REPEAT          ; Beginning of REPEAT..UNTIL() expression
IF(IN.1=b1)     ; Check for input #1 becoming active
VAR1=VAR1+10    ; If it does, increase variable 1 by 10
VAR2=1          ; Variable 2 is used as a flag
NIF             ; End IF statement
UNTIL(MOV=b0 OR VAR2=5)    ; Exit REPEAT loop if variable 2 equals 5 or if
                ; motion is complete on axis 1
VAR2=0          ; Reset flag value, variable 2 = 0
LN              ; End loop
COMEXC0         ; Disable continuous command mode
```

On-the-fly Velocity, Acceleration and Deceleration Change Example:

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

COMEXK Continue Execution on Kill

Type	Command Buffer Control
Syntax	<!>COMEXK<b>
Units	b = 0, 1 or X
Range	0 = Disable, 1 = Enable, X = don't change
Default	0
Response	COMEXK: *COMEXK0
See Also	COMEXC, COMEXL, COMEXP, COMEXS, ERROR, INFNC, K, <ctrl>K

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	n/a
615n	n/a
620n	n/a
625n	n/a
6270	n/a

This command determines whether the commands following a Kill (K) command in a block write will be saved after the (K) command is processed. Upon receiving a (K) command, or an external kill input (INFNCi-C), all commands in the command buffer are eliminated. If there are any other commands contained within the data block during the Kill (K) command, these commands will also be eliminated from the command buffer, unless Continue Execution on Kill (COMEXK) is enabled. This also holds true when a Kill input is received.

Example:

```
COMEXK1         ; Save block write data upon a kill input or kill command
```

COMEXL Continue Execution on Limit

		Product	Rev
Type	Command Buffer Control	AT6n00	1.0
Syntax	<!><@><a>COMEXL<b><b><b><b>	AT6n50	1.0
Units	b = 0, 1 or X	610n	4.0
Range	0 = Disable, 1 = Enable, X = don't change	615n	1.0
Default	0	620n	1.0
Response	COMEXL: *COMEXL0000 1COMEXL: *1COMEXL0	625n	1.0
See Also	COMEXC, COMEXK, COMEXP, COMEXS, ERROR, LH, LHLVL, LS	6270	1.0

This command determines whether the command buffer will be saved upon hitting an end-of-travel limit (LH), or a soft limit LS). If save command buffer on limit is enabled (COMEXL1111), then all commands following the command currently being executed will remain in the command buffer when a limit is hit. If save command buffer on limit is disabled (COMEXLØØØØ), then every command in the buffer will be discarded, and program execution will be terminated.

Example:

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

COMEXP Continue Execution on In Position

		Product	Rev
Type	Command Buffer Control	AT6n00	1.0
Syntax	<!><@><a>COMEXP<b><b><b><b>	OEM-AT6n00	n/a
Units	b = 0, 1 or X	AT6n50	n/a
Range	0 = Disable, 1 = Enable, X = don't change	610n	n/a
Default	0	615n	n/a
Response	COMEXP: *COMEXP0000 1COMEXP: *1COMEXP0	6200	1.0
See Also	[AS], COMEXC, COMEXK, COMEXL, COMEXS, INFNC, S, TAS	6201	n/a
		625n	n/a
		6270	n/a

This command determines whether the command processing will pause until the *in position* signal is received (via the motor/drive connector). When enabled (COMEXP1), command processing is paused until the *in position* input is active. Once active, command processing continues.

If disabled (COMEXPØ), command processing will continue as soon as the 6000 Series product finishes commanding the desired position, even if the motor/drive combination is not *in position*.

To be *in position*, the motor/drive must be within its maximum deadband for a fixed period of time, and activate its own in position output after this period of time.

Bit 20 of the axis status register ([AS] and TAS) reports the *in position* status.

Example:

```
COMEXP1111      ; Command processing will wait until all axes are in position
```

COMEXR Continue Motion on Pause/Continue Input

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

Product	Rev
AT6n00	1.4
OEM-AT6n00	n/a
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.5
625n	1.0
6270	1.0

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

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

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

Example:

```
COMEXR1          ; Allow both motion and program execution to be paused upon
                 ; receiving a pause input
INFNC1-E         ; Define programmable input #1 as a pause/continue input
INFEN1           ; Enable input functions
```

COMEXS Continue Execution on Stop

Type	Command Buffer Control
Syntax	<!>COMEXS<i>
Units	i = function identifier
Range	0, 1, or 2
Default	0
Response	COMEXS: *COMEXS0
See Also	COMEXC, COMEXK, COMEXL, COMEXP, COMEXR, INFNC, S

Product	Rev
AT6n00	1.4
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.5
625n	1.0
6270	1.0

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

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

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

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

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

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

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

[COS()] Cosine

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

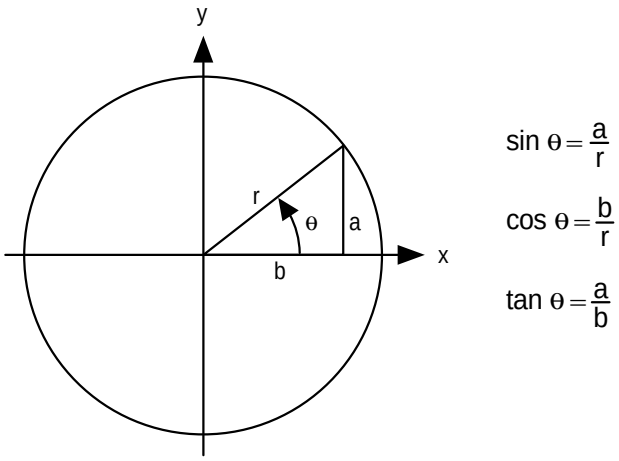
Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

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

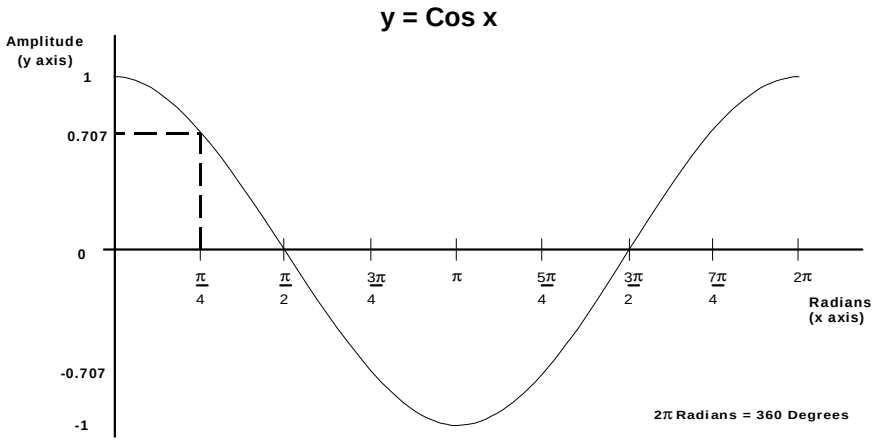
$\cos\theta = \frac{b}{r}$ (see illustration at right)

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

360° = 2π radians.



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



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

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

D — Distance

Type	Motion
Syntax	<!><@><a>D<r>,<r>,<r>,<r>
Units	r = distance units (scalable)
Range	0.00000 - ±999,999,999. Steppers: Max. distance depends on PULSE setting
Default	25000 (AT6n00, 620n, & 610n); 4000 (AT6n50 & 625n); 1000 (6270); 4096 (615n)
Response	D: *D+25000,+25000,+25000,+25000 1D: *1D+25000
See Also	[D], GO, MA, MC, PSET, PULSE, SCLD, TSTAT

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

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

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

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

ON-THE-FLY CHANGES (as of revision 4.0): You can change distance *on the fly* (while motion is in progress) in two ways. One way is to send an immediate distance command (!D) followed by an immediate go command (!GO). The other way is to enable the continuous command execution mode (COMEXC1) and execute a buffered distance command (D) followed by a buffered go command (GO).

In **stepper systems** with scaling disabled (SCALEØ), all distance values entered are in either motor steps or encoder steps, depending on the state of the Encoder/Motor Step Mode (ENC) command.

In **servo systems** with scaling disabled (SCALEØ), all distance values are in encoder, resolver, LDT or ANI counts.

The maximum distance in stepper systems is determined by the PULSE command setting.

Pulse Width (PULSE) Setting	Maximum Distance Per Move	Maximum Velocity
DEFAULT – 0.3 µs	419,430,000	1.6 MHz
0.5 µs	262,140,000	1.0 MHz
Use for Compumotor's Z and DB Drives – 1.0 µs	131,070,000	500 KHz
2.0 µs	65,535,000	250 KHz
5.0 µs	26,214,000	100 KHz
10.0 µs	13,107,000	50 KHz
16.0 µs	8,191,000	35 KHz
20.0 µs	6,553,000	25 KHz

Scaling: If scaling is enabled (SCALE1), the distance (D) value is internally multiplied by the distance scale factor (SCLD) to obtain a distance value in motor steps or feedback device (encoder, resolver, LDT, or ANI) steps for the motion trajectory calculations.

As the distance scaling factor (SCLD) changes, the resolution of the distance (D) command and the number of positions to the right of the decimal point also change (see table below). A distance value with greater resolution than allowed will be truncated. For example, if scaling is set to SCLD25ØØØ, the D1.99999 command would be truncated to D1.9999.

SCLD (steps/unit)	Distance Resolution (units)	Distance Range (units)	Decimal Places
1 - 9	1	0 - ±999,999,999	0
10 - 99	0.1	0.0 - ±99,999,999.9	1
100 - 999	0.01	0.00 - ±9,999,999.99	2
1000 - 9999	0.001	0.000 - ±999,999.999	3
10000 - 99999	0.0001	0.0000 - ±99,999.9999	4
100000 - 999999	0.00001	0.00000 - ±9999.99999	5

The distance scaling factor should always be enabled and specified prior to entering any distance values, because the SCLD command modifies the current distance value to accommodate the new scaling factor.

NOTE — FRACTIONAL STEP TRUNCATION — NOTE

There is one consideration that must be taken into account when using the distance scale factor (SCLD), but only while operating in the preset positioning mode (MCØ). When the distance scaling factor and the distance value are multiplied, a fraction of one step may possibly be left over. This fraction is truncated when the distance value is used in the move algorithm. This truncation error can accumulate over a period of time, when performing incremental moves continuously in the same direction. To eliminate this truncation problem set the distance scale factor (SCLD) to 1, or a multiple of 10.

Example:

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

[D] Distance Assignment

		Product	Rev
Type	Assignment or Comparison	AT6n00	1.0
Syntax	See below	AT6n50	1.0
Units	distance units (scalable)	610n	4.0
Range	0.00000 - ±999,999,999	615n	1.0
Default	n/a	620n	1.0
Response	n/a	625n	1.0
See Also	D, GO, MA, MC, PSET, SCLD	6270	1.0

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

Syntax: VARn=aD where n is the variable number, and a is the axis number, or [D] can be used in an expression such as IF(1D<25000). When assigning the distance value to a variable, an axis specifier must always precede the D command (e.g., VAR1=1D) or it will default to axis 1. When making a comparison to the programmed distance, an axis specifier must also be used (e.g., IF(1D<20000)). The D value used in any comparison, or in any assignment statement is the programmed D value. If the actual position information is required, refer to the PM command for steppers, or the ANI, LDT, or PE commands for servos.

If scaling is enabled, the distance value is scaled by the SCLD command. If you are using a servo controller with ANI feedback, the distance value is scaled by the SCLANI value.

Example:

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

[DAC] Value of DAC Output

		Product	Rev
Type	Assignment or Comparison	AT6n00	n/a
Syntax	See below	AT6n50	1.0
Units	Volts	610n	n/a
Range	-10.000 to +10.000	615n	1.0
Default	n/a	620n	n/a
Response	n/a	625n	3.0
See Also	DACLIM, DACMIN, SOFFS, SOFFSN, TDAC	6270	1.0

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

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

Example:

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

DACTDP Active Damping

Type	Drive Configuration
Syntax	<!><@><a>DACTDP<i>
Units	see table below
Range	0-15 (see inertia table below) 0 disables active damping
Default	0
Response	DACTDP *DACTDP0 1DACTDP *1DACTDP0
See Also	DAUTOS, DAREN, DELVIS, DMTIND, DMTSTT, DWAVEF

Product	Rev
AT6n00	n/a
AT6n50	n/a
610n	4.0
615n	n/a
620n	n/a
625n	n/a
6270	n/a

The Active Damping (DACTDP) command is used with the 610n to configure active damping for a specific motor and load. If the rotor rings or vibrates, the active damping circuit in the 610n will detect the corresponding error in rotor position. It will then modify the motor current command to damp the ringing. Active damping works at speeds greater than three revolutions per second.

To be fully effective, the active damping circuit requires the proper system parameters to be set (motor inductance, motor static torque, and system inertia). To do this, first calculate your maximum system inertia (including rotor). Then consult the table of inertia ranges below. Find the value for <i> that corresponds to your maximum system inertia (not to exceed). Refer to the *ZETA610n Installation Guide* for the full procedure for configuring active damping. (Table is for reference only.)

NOTE: Motor Inductance (DMTIND) and Motor Static Torque (DMTSTT) must be set properly for the values in this table to apply. If active damping is enabled above 3 rps, anti-resonance (DAREN) is automatically overridden. Once active damping is disabled, anti-resonance will automatically resume.

<i>	Total Inertia kg-cm^2	Total Inertia kg-m^{2}x 10^{-6}	Total Inertia oz-in^2
15	0.088 – 0.205	8.8 – 20.5	0.481 – 1.121
14	0.205 – 0.572	20.5 – 57.2	1.121 – 3.144
13	0.572 – 1.069	57.2 – 106.9	3.127 – 5.845
12	1.069 – 1.754	106.9 – 175.4	5.845 – 9.590
11	1.754 – 2.727	175.4 – 272.7	9.590 – 14.910
10	2.727 – 3.715	272.7 – 371.5	14.910 – 20.312
9	3.715 – 5.020	371.5 – 502.0	20.312 – 27.447
8	5.020 – 6.275	502.0 – 627.5	27.447 – 34.308
7	6.275 – 8.045	627.5 – 804.5	34.308 – 43.986
6	8.045 – 9.595	804.5 – 959.5	43.986 – 52.460
5	9.595 – 11.76	959.5 – 1176	52.460 – 64.297
4	11.76 – 14.25	1176 – 1425	64.297 – 77.884
3	14.25 – 15.90	1425 – 1590	77.884 – 86.905
2	15.90 – 17.77	1590 – 1777	86.905 – 97.129
1	17.77 – 20.57	1777 – 2057	97.129 – 112.465
0	Active Damping Disabled		

Active Damping Command Settings & Corresponding Inertia Ranges

The following example shows a sample start-up program to set active damping (> 3 rps) and electronic viscosity (< 3 rps), where the optimal system inertia and electronic viscosity settings are already known.

Example:

```
DEFstart1       ; Begin definition of start1
DAREN0          ; Turn off Anti-Resonance
DMTIND3         ; Set Motor Inductance to the range for a
                ; 57-51P ZETA Motor (5.03 - 10.30 mH)
DMTSTT1         ; Set Motor Static Torque to the range for a
                ; 57-51 ZETA Motor (0.26 - 0.72 N-m)
DACTDP7         ; Set Active Damping for system inertia of
                ; 6.275 - 8.045 kg-cm2 (optimal setting previously determined)
DELVIS3         ; Set Electronic Viscosity
                ; (optimal setting previously determined)
END             ; End program definition
```

DACLIM Digital-to-Analog Converter (DAC) Limit

		Product	Rev
Type	Servo	AT6n00	n/a
Syntax	<!><@><a>DACLIM<r>,<r>,<r>,<r>	AT6n50	1.0
Units	r = volts	610n	n/a
Range	0.000 to 10.000	615n	1.0
Default	10.000	620n	n/a
Response	DACLIM: *DACLIM10.00,10.00,10.00,10.00 1DACLIM: *1DACLIM10.00	625n	1.0
See Also	[DAC], DACMIN, SOFFS, SOFFSN, TDAC	6270	1.0

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

Example:

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

DACMIN Minimum DAC Output Voltage

		Product	Rev
Type	Servo	AT6n00	n/a
Syntax	<!><@><a>DACMIN<r>,<r>	AT6n50	n/a
Units	r = volts	610n	n/a
Range	-10.000 to +10.000	615n	n/a
Default	-10.000	620n	n/a
Response	DACMIN *DACMIN-10.000,-10.000 1DACMIN *1DACMIN-10.000	625n	n/a
See Also	[DAC], DACLIM, SOFFS, SOFFSN, TDAC	6270	3.0

In addition to the defining the maximum DAC voltage limit with the DACLIM command, you may also define a minimum DAC voltage limit with the DACMIN command. In so doing, you limit the DAC output to the range between DACMIN and DACLIM. You may not set the DACMIN value greater than the DACLIM value (default DACLIM value is +10.000 volts).

For example (2-axis controller), if on both axes you wish to prevent motion in the negative direction and limit the maximum voltage to 8.5V, use the DACMINØ,Ø and DACLIM8.5,8.5 commands. The DACMINØ,Ø command sets the minimum DAC output to zero volts, which effectively prevents the controller from commanding motion in the negative direction. The DACLIM8.5,8.5 command sets the maximum DAC output to 8.5V.

By default the DACMIN value tracks (but with a reverse sign) the value of DACLIM, until a DACMIN command is executed. After the DACMIN command is executed, DACMIN and DACLIM values must be entered separately. To re-establish the default mode where DACMIN tracks DACLIM, issue the DACMIN command with only a minus sign (-) in the command field for the affected axis (e.g., DACMIN,- restores axis 2 to the default mode).

Example:

```
COMEXC1          ; Enable continuous command execution mode
SFB3             ; Select LDT feedback
MC1              ; Select continuous positioning mode
A5               ; Set acceleration to 5 units/sec/sec
V5               ; Set velocity to 5 units/sec
GO1              ; Begin moving
WHILE(1ANI<5)    ; While ANI voltage is less than 5V, continue moving
NWHILE
S                ; Stop motion (occurs when ANI voltage is no longer < 5V)
DACMIN0          ; Do not allow reverse movement (and hold ANI voltage to >= 5V)
```

4-20mA Control Example *(6270 only)***:**

If are using a 4-20mA control loop, set the analog output jumpers to operate at ±20mA (instructions provided in the 6270 installation/user guide). Then issue these setup commands listed below. (Note that

when using ±20mA output, you need to use the 2mA/volt equation to ascertain the proper *voltage* value to enter in the DACLIM, DACMIN, and SOFFS commands).

Example:

```
DACLIM10        ; Set DAC maximum limit to +20mA (20mA ÷ 2mA/V = 10V)
DACMIN2         ; Set DAC minimum limit to +4mA (4mA ÷ 2mA/V = 2V)
SOFFS6          ; Set offset analog output to the mid-range value of +12mA (12mA
                ÷ 2mA/V = 6V)
```

DAREN — Anti-Resonance

		Product	Rev
Type	Drive Configuration	AT6n00	n/a
Syntax	<!><@><a>DAREN<b>	AT6n50	n/a
Units	n/a	610n	4.0
Range	0 (disable), 1 (enable), x (don't change)	615n	n/a
Default	1	620n	n/a
Response	DAREN *DAREN1 1DAREN *1DAREN1	625n	n/a
See Also	DAUTOS, DACTDP, DELVIS, DMTIND, DMTSTT, DWAVEF	6270	n/a

The Anti-Resonance (DAREN) command is used with the 610n to provide aggressive and effective damping. When enabled with the DAREN1 command, the anti-resonance circuitry in the 610n minimizes rotor position error, without knowledge about the system — whether the motor is large or small, or the system inertia is high or low. Anti-resonance works at speeds greater than three revolutions per second.

NOTE: If active damping is enabled, anti-resonance is automatically overridden. Once active damping is disabled, anti-resonance will automatically resume.

[DAT] — Data Assignment

		Product	Rev
Type	Data Storage	AT6n00	1.0
Syntax	DATi	AT6n50	1.0
Units	i = data program #	610n	4.0
Range	1 - 50	615n	1.0
Default	n/a	620n	1.0
Response	n/a	625n	1.0
See Also	DATA, [DATP], DATPTR, DATRST, DATTCH	6270	1.0

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

Syntax: VARn=DATi where n is the variable number, and i is the data program number, or [DATx] can be used as a command argument such as A(DAT1),5,4,10

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

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

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

DATA — Data Statement

		Product	Rev
Type	Data Storage	AT6n00	1.0
Syntax	<!>DATA=<r>,<r>,<r>,<r>	AT6n50	1.0
Units	r = data value	610n	4.0
Range	±999,999,999.99999999	615n	1.0
Default	n/a	620n	1.0
Response	n/a	625n	1.0
See Also	[DAT], [DATP], DATPTR, DATRST, DATTCH, MEMORY	6270	1.0

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

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

[DATP] — Data Program

		Product	Rev
Type	Data Storage	AT6n00	1.0
Syntax	DATPi	AT6n50	1.0
Units	i = data program #	610n	4.0
Range	1 - 50	615n	1.0
Default	n/a	620n	1.0
Response	n/a	625n	1.0
See Also	[DAT], DATA, DATPTR, DATRST, DATSIZ, DATTCH, MEMORY	6270	1.0

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

Example:

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

DATPTR Set Data Pointer

		Product	Rev
Type	Data Storage	AT6n00	2.2
Syntax	<!>DATPTRi,i,i	AT6n50	1.0
Units	n/a	610n	4.0
Range	1st i = program # 1 to 50 2nd i = data element # 1 to 6500 3rd i = increment setting of 1 to 100	615n	1.0
		620n	2.4
		625n	3.0
Default	1,1,1	6270	1.0
Response	n/a		
See Also	[DAT], DATA, [DATP], DATSIZ, DATTCH, [DPTR], TDPTR		

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

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

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

As an example, suppose data program #1 (DATP1) is configured to hold 15 data elements (DATSIZ1,15), the data pointer is configured to start at the first data element and increment 1 data element after every DATTCH value is stored (DATPTR1,1,1), and the values of numeric variables #1 through #4 are already assigned (VAR1=2, VAR2=4, VAR3=8, VAR4=64). If you then enter the DATTCH1,2,3,4 command, the values of VAR1 through VAR4 will be assigned respectively to the first four data elements in the data program and the pointer will stop at data element #5. The response to the TPROG DATP1 command would be as depicted below (the text is highlighted to illustrate the location of the data pointer after the DATTCH1,2,3,4 command is executed). The response to the TDPTR command would be *TDPTR1,5,1.

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

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

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

Example: (See Also: DATSIZ command)

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

DATRST Reset Data Pointer

		Product	Rev
Type	Data Storage	AT6n00	1.0
Syntax	<!>DATRST<i>,<i>	AT6n50	1.0
Units	n/a	610n	4.0
Range	1st i = program # 1 to 50, 2nd i = data element # 1 to 6500	615n	1.0
Default	n/a	620n	1.0
Response	n/a	625n	1.0
See Also	[DAT], DATA, [DATP]	6270	1.0

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

Example:

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

DATSIZ Data Program Size

		Product	Rev
Type	Data Storage	AT6n00	2.2
Syntax	<!>DATSIZi<,i>	AT6n50	1.0
Units	n/a	610n	4.0
Range	1st i = program # 0 - 50 (0 = disable) 2nd i = data element # 1 - 6500	615n	1.0
Default	0,1	620n	2.4
Response	n/a	625n	3.0
See Also	[DAT], DATPTR, [DATP], DATTCH	6270	1.0

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

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

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

Each data statement, which contains four data elements, uses 39 bytes of memory. This amount of memory is subtracted from the memory allocated for user programs (see MEMORY command). Use the TDIR command to determine the amount of remaining memory for user program storage.

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

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

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

1. Issue the DATSIZ command to create (or recall) the data program.
2. Store variable data (e.g., position, acceleration, velocity, etc.) to numeric variables (VAR).
3. Use DATTCH commands to store the data from the numeric variables into the data program. You can use the data pointer (DATPTR) command to select any data element in the data program, and to determine the number by which the pointer increments after each value from the DATTCH command is stored. *NOTE: If the DATTCH command is issued without having issued the DATSIZ command, an error will result.*
4. Use the DAT commands to read the stored data from the data program into the variable parameters of your motion program. You can use the DATPTR command to select any data element in the data program, and to determine the number by which the pointer increments after each DAT command.

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

Example:

```
DEL DATP5         ; Delete existing data program #5 (DATP5)
DATSIZ5,200       ; Create data program #5 (DATP5) with 200 data elements
DEF TEACH         ; Begin definition of program called TEACH
COMEXC0           ; Disable continuous command execution mode
MA1111            ; Enable the absolute positioning mode for all axes
HOM1111           ; Home all axes (absolute position counter set to zero after homing)
DATPTR5,1,1       ; Set data pointer to data element #1 in DATP5, and increment the
                  ; pointer by one element after every DATTCH value or DAT command
REPEAT            ; Set up a loop for teaching the positions
JOY1111           ; Enable joystick mode on all axes so that you can start moving the
                  ; axes into position with the joystick. Command processing stops
                  ; here until you activate the joystick release input to disable the
                  ; joystick mode and execute the rest of the commands in the
                  ; repeat/until loop (assign the motor positions to the variables and
                  ; then store the positions from the variables to the data program).
VAR1=1PM          ; Store the current position of axis #1 in variable #1
VAR2=2PM          ; Store the current position of axis #2 in variable #2
VAR3=3PM          ; Store the current position of axis #3 in variable #3
VAR4=4PM          ; Store the current position of axis #4 in variable #4
DATTCH1,2,3,4     ; Store variables #1 - #4 into consecutive data elements
WAIT(INO.5=b1)    ; Wait for the joystick release input to be de-activated
UNTIL(DPTR=1)     ; Repeat loop until the data pointer wraps around to data element #1
HOM1111           ; Home all axes (absolute position counter set to zero after homing)
DATPTR5,1,1       ; Set data pointer to data element #1, read one data element at a time
REPEAT            ; Set up a repeat/until loop to read all data elements
D(DAT5),(DAT5),(DAT5),(DAT5)      ; Read position data from the data program to the
                  ; distance command
GO1111            ; Make the move to the positions that were taught
T.2               ; Wait 0.2 seconds
UNTIL(DPTR=1)     ; Repeat loop until the data pointer wraps around to data element #1
END               ; End definition of program called TEACH
```

DATTCH Data Teach

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

Product	Rev
AT6n00	2.2
AT6n50	1.0
610n	4.0
615n	1.0
620n	2.4
625n	3.0
6270	1.0

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

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

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

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

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

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

DAUTOS Auto Current Standby

Type	Drive Configuration
Syntax	<!><@><a>DAUTOS<b>
Units	n/a
Range	0 (disable), 1 (enable), x (don't change)
Default	0
Response	DAUTOS *DAUTOS0 1DAUTOS *1DAUTOS0
See Also	DAREN, DACTDP, DELVIS, DMTIND, DMTSTT, DWAVEF

Product	Rev
AT6n00	n/a
AT6n50	n/a
610n	4.0
615n	n/a
620n	n/a
625n	n/a
6270	n/a

The Auto Current Standby (DAUTOS) command is used to allow the motor to cool when it is not moving. When the automatic current standby function is enabled (DAUTOS1), the 610n reduces motor current by 50% if a step pulse is not issued from the controller to the drive (within the 610n) for one second. Full current is restored upon the first step pulse that the drive receives.

WARNING: Motor torque is reduced when motor current is reduced.

DCLEAR Clear Display

		Product	Rev
Type	Display (RP240) Interface		
Syntax	<!>DCLEARi	AT6n00	n/a
Units	n/a	AT6n50	n/a
Range	i = 0 (clear all lines), 1 (clear line 1), or 2 (clear line 2)	610n	4.0
		615n	1.0
Default	n/a	620n	1.0
Response	n/a	625n	1.0
See Also	DLED, DPASS, DPCUR, DVAR, DWRITE	6270	1.0

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

DEF Begin Program/Subroutine/Path Definition

		Product	Rev
Type	Program or Subroutine Definition		
Syntax	<!>DEF<t>	AT6n00	1.0
Units	t = alpha text string (name of a program)	AT6n50	1.0
Range	text string of 6 characters or less	610n	4.0
Default	n/a	615n	1.0
Response	n/a	620n	1.0
See Also	$, DEL, END, ERASE, GOBUF, GOSUB, GOTO, MEMORY, PCOMP, PRUN, RUN, [SS], TDIR, TMEM, TPROG, TSS, TSTAT	625n	1.0
		6270	1.0

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

All programs are stored in a binary fashion within the 6000 Series products. A program transferred back out (TPROG) after it has been defined (DEF), may not look identical to the program defined. However, the program is functionally identical.

NOTE

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

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

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

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

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

Example:

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

DEL — Delete a Program/Subroutine/Path

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

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

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

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

Example:

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

DELVIS — Electronic Viscosity

Type	Drive Configuration
Syntax	<!><@><a>DELVIS<i>
Units	n/a
Range	0-7 0 disables Electronic Viscosity
Default	0
Response	DELVIS *DELVIS0 1DELVIS *1DELVIS0
See Also	DAREN, DACTDP, DAUTOS, DMTIND, DMTSTT, DWAVEF

Product	Rev
AT6n00	n/a
AT6n50	n/a
610n	4.0
615n	n/a
620n	n/a
625n	n/a
6270	n/a

The Electronic Viscosity (DELVIS) command is used to activate circuitry in the 610n that provides damping from rest at speeds up to three revolutions per second, and to set the amount of required viscosity. The effect on the motor is as if there were a viscous drag on the rotor. At the end of a move, oscillations are damped, and the rotor quickly settles at the commanded velocity. During moves below 3 rps, DELVIS significantly reduces low speed velocity ripple.

The circuitry controlled by the DELVIS command imposes viscosity on the system, but has no feedback loop to monitor the effect of the viscosity. DELVIS keeps the amount of viscosity the same, regardless of the response of the system.

Refer to the *ZETA610n Installation Guide* for the full procedure for configuring electronic viscosity. Also see the example for the DACTDP command.

DJOG — Enable RP240 Jog Mode

Type	Display (RP240) Interface
Syntax	<!>DJOG<b>
Units	b = 0 or 1
Range	0 = disable, 1 = enable
Default	n/a
Response	DJOG: *DJOG1
See Also	JOG, JOGA, JOGAA, JOGAD, JOGADA, JOGVH, JOGVL

Product	Rev
AT6n00	n/a
AT6n50	n/a
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

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

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

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

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

DLED — Turn RP240 Display LEDs On/Off

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

Product	Rev
AT6n00	n/a
AT6n50	n/a
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

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

Example:

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

DMTIND Motor Inductance

		Product	Rev
Type	Drive Configuration	AT6n00	n/a
Syntax	<!><@><a>DMTIND<i>	AT6n50	n/a
Units	n/a	610n	4.0
Range	1-4	615n	n/a
Default	1	620n	n/a
Response	DMTIND *DMTIND1 1DMTIND *1DMTIND1	625n	n/a
See Also	DAREN, DACTDP, DAUTOS, DELVIS, DMTSTT, DWAVEF	6270	n/a

The Motor Inductance (DMTIND) command is used to indicate motor inductance for the active damping circuit in the 610n. This command should be set according to your motor's *large-signal* inductance. The table below shows the large-signal inductance range that corresponds to each of the four settings.

<i>	Inductance Range (mH)	ZETA Motor
1	20.08 & Greater	57-51**S**, 57-83**S**, 57-102**S**
2	10.31 – 20.07	83-62**S**, 83-93**S**, 83-135**S**
3	5.03 – 10.30	57-51**P**, 57-83**P**, 57-102**P**
4	less than 5.02	83-62**P**, 83-93**P**, 83-135**P**

Motor Inductance Command Settings

Small signal inductance is the value read on an ordinary inductance bridge or meter. Large signal inductance is found by measuring the actual generator AC flux leakage and generator short circuit current under dynamic conditions.

If you only have the small-signal inductance available, use this formula to approximate large-signal inductance:

$$\textit{small-signal inductance} * 1.5 \approx \textit{large signal inductance}$$

NOTE: If active damping (DACTDP) is disabled, DMTIND is ignored by the drive and the command is inactive. Refer to the *ZETA610n Installation Guide* for additional information about how the DMTIND and DACTDP commands work together. Also see the example for the DACTDP command.

The DMTIND command value is automatically saved in battery-backed RAM.

DMTSTT Motor Static Torque

		Product	Rev
Type	Drive Configuration	AT6n00	n/a
Syntax	<!><@><a>DMTSTT<i>	AT6n50	n/a
Units	n/a	610n	4.0
Range	1-4	615n	n/a
Default	1	620n	n/a
Response	DMTSTT *DMTSTT1 1DMTSTT *1DMTSTT1	625n	n/a
See Also	DAREN, DACTDP, DAUTOS, DMTIND, DELVIS, DWAVEF	6270	n/a

The Motor Static Torque (DMTSTT) command is used to indicate the motor's static torque for the active damping circuit in the 610n. The table shows the range of static torque that corresponds to each of the four settings.

<i>	Static Torque (N-m)	Static Torque (oz-in)	ZETA Motor
1	0.26 – 0.72	36 – 100	57-51, 57-83
2	0.73 – 1.41	101 – 200	57-102, 83-62
3	1.42 – 2.33	201 – 330	83-93, 83-135**P**
4	2.34 – 3.48	331 – 492	83-135**S**

Motor Static Torque Command Settings

NOTE: If active damping (DACTDP) is disabled, DMTSTT is ignored by the drive and the command is inactive. Refer to the *ZETA610n Installation Guide* for additional information about how the DMTSTT and DACTDP commands work together. Also see the example for the DACTDP command.

The DMTSTT command value is automatically saved in battery-backed RAM.

DPASS — Change RP240 Password

Type	Display (RP240) Interface
Syntax	<!>DPASS<i>
Units	i = integer of up to 4 characters
Range	1 - 9999
Default	For the 6200: DPASS6200 For the 6201: DPASS6200 For the 6250: DPASS6250 For the 6270: DPASS6270 For the ZETA610n: DPASS6104 For the APEX615n: DPASS6150
Response	DPASS: *DPASS6200
See Also	DCLEAR, DLED, DPCUR, DVAR, DWRITE

Product	Rev
AT6n00	n/a
AT6n50	n/a
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

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

Example:

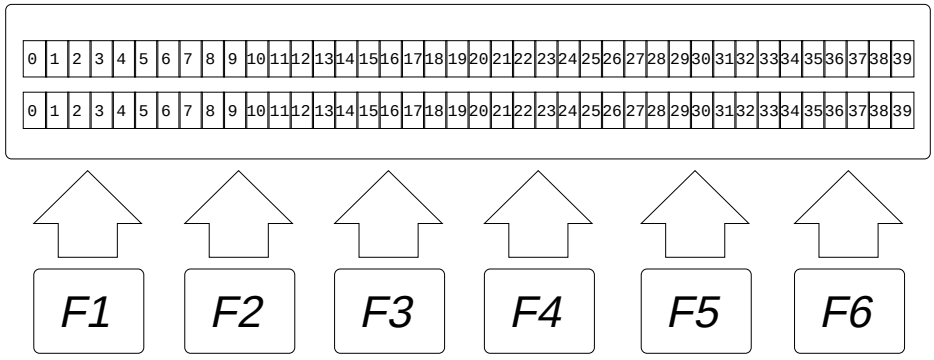
```
DPASS1234       ; New password = 1234
```

DPCUR — Position Cursor

Type	Display (RP240) Interface
Syntax	<!>DPCURi,i
Units	1st i = line number, 2nd i = column
Range	line number = 1 or 2, column = 0 - 39
Default	n/a
Response	n/a
See Also	DCLEAR, DLED, DPASS, DREADI, DVAR, DWRITE

Product	Rev
AT6n00	n/a
AT6n50	n/a
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Position Cursor (DPCUR) command changes the location of the cursor on the RP240 display. The RP240 lines are numbered from top to bottom, 1 to 2. The columns are numbered left to right, 0 to 39.



Example:

```
DPCUR2,15       ; Position cursor on line 2, column 15
```

[DPTR] Data Pointer Location

Type	Data Storage; Assignment or Comparison
Syntax	see below
Units	n/a
Range	n/a
Default	n/a
Response	n/a
See Also	[DAT], DATA, [DATP], DATPTR, DATSIZ, TDPTR

Product	Rev
AT6n00	2.2
AT6n50	1.0
610n	4.0
615n	1.0
620n	2/4
625n	3.0
6270	1.0

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

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

Example:

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

[DREAD] Read RP240 Data

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

Product	Rev
AT6n00	n/a
AT6n50	n/a
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

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

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

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

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

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

[DREADF] Read RP240 Function Key

Type	Display (RP240) Interface
Syntax	See below
Units	n/a
Range	n/a
Default	n/a
Response	n/a
See Also	[DREAD], DREADI, DVAR, DWRITE, [SS], TSS, VAR

Product	Rev
AT6n00	n/a
AT6n50	n/a
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

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

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

Example:

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

DREADI RP240 Data Read Immediate Mode

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

Product	Rev
AT6n00	n/a
AT6n50	n/a
610n	4.0
615n	1.0
620n	2.1
625n	1.1
6270	1.0

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

NOTES

- While in the Data Read Immediate Mode (DREADI1), data is read into **numeric variables only** (e.g., A(DREAD) or V(DREAD) will not be valid).
- This feature is not designed to be used in conjunction with the RP240's standard menus (see *6000 Series Programmer's Guide* for menu structure); the **RUN**, **JOG**, and **DJOG** menus will disable the DREADI mode.
- Do not assign the same variable to read numeric data **and** function key data—pick only one.

Programming Examples on next page ...

Simple Numeric Data Entry (example):

```
VAR1=25000      ; Initialize variable #1
DCLEAR0         ; Clear entire RP240 display
DWRITE"ENTER VALUE>" ; Send message to RP240 display starting at location 1,0
DREADI1         ; Enable RP240 data read immediate mode
VAR1=DREAD      ; Set variable #1 (VAR1) to receive data entered on the RP240.
                ; Current VAR1 data will be displayed at cursor location 1,30
                ; (fixed). New data will be displayed at current cursor location
                ; as defined by the previous DCLEAR, DWRITE and DPCUR commands—
                ; this is the home cursor location for subsequent data entries.
L77             ; Start loop of 77 repetitions
D(VAR1)         ; Set distance equal to the current (last entered) RP240 data
GO1             ; Initiate move on axis one
LN              ; End loop
DREADI0         ; Exit RP240 data read immediate mode
; As the loop is running, the user may enter in a new distance value
; (which must be terminated with the ENTER key) via the RP240 numeric keypad.
; The numeric keystrokes cause the digits to be displayed on the RP240
; starting at the home cursor location (see VAR1=DREAD description in the
; example above). When the ENTER key is pressed, the variable is updated;
; the most significant 10 digits (total, including sign & decimal point
; if appropriate) of this variable are displayed at cursor location 1,30;
; and then the data entry field (starting at home) is cleared.
; The 6000 controller is ready to accept new data.
```

Numeric Data & Function Key Entry (example):

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

Multiple Numeric Data Entry (example):

```
VAR2=0          ; Initialize variable #2 (VAR2)
VAR3=99         ; Initialize variable #3 (VAR3)
VAR4=10         ; Initialize variable #4 (VAR4)
VAR5=25000      ; Initialize variable #5 (VAR5)
DCLEAR0         ; Clear the entire RP240 display
DPCUR2,0        ; Place RP240 cursor on line 2, column 0 (bottom left corner of
                ; display)
DWRITE" ACCEL VEL DIST"  ; Send message to RP240 display starting at location 2,0
DREADI1         ; Enable RP240 data read immediate mode
VAR2=DREADF     ; VAR2 will capture function key entries (0 - 6)
L               ; Begin loop
IF(VAR2<>0)     ; If a new function key is pressed, do the following code:
DCLEAR1         ; Clear line one of the RP240 display (top line)
```

```
IF(VAR2=1)      ; If function key 1 is pressed, do the IF statement
                ; (input acceleration)
DWRITE"ENTER ACCEL VALUE>" ; Send message to RP240 display starting at location 1,0
VAR3=DREAD      ; Set VAR3 equal to the numeric data entered on the RP240's keypad
NIF             ; End IF statement
IF(VAR2=2)      ; If function key 2 is pressed, do the IF statement (input velocity)
DWRITE"ENTER VEL VALUE>" ; Send message to RP240 display starting at location 1,0
VAR4=DREAD      ; Set VAR4 equal to the numeric data entered on the RP240's keypad
NIF             ; End IF statement
IF(VAR2=3)      ; If function key 3 is pressed, do the IF statement (input distance)
DWRITE"ENTER DIST VALUE>" ; Send message to RP240 display starting at location 1,0
VAR5=DREAD      ; Set VAR5 equal to the numeric data entered on the RP240's keypad
NIF             ; End IF statement
VAR2=0          ; Prohibit repeated execution of this code
VAR2=DREADF     ; Re-enable VAR2 to capture new function key entry
NIF             ; End IF statement
A(VAR3)         ; Set acceleration equal to the numeric value of VAR3
V(VAR4)         ; Set velocity equal to the numeric value of VAR4
D(VAR5)         ; Set distance equal to the numeric value of VAR5
GO1             ; Initiate the move on axis one
LN              ; End loop
; As the loop is running, the user may select among the three variables he wants
; to enter data into. These three variables correspond with acceleration,
; velocity, and distance. Each time the function key variable changes from 0
; (to 1, 2 or 3), then a new message is displayed and the VARi=DREAD command
; will put the current value of that variable in location 1,30 (upper right hand
; corner of the display). For example, the user can choose VEL (F2) and then
; repeatedly change VAR4 by entering a value on the RP240 numeric keypad and
; pressing the ENTER key. Each time through the loop, the VAR4 data is loaded
; into the V command.
```

DRES — Drive Resolution

Type	Drive Configuration
Syntax	<!><@><a>DRES<i>,<i>,<i>,<i>
Units	i = steps/rev
Range	200 - 1024000
Default	25000
Response	DRES: *DRES25000,25000,25000,25000 1DRES: *1DRES25000
See Also	DRFLVL, DRIVE, ENC, ERES, PCOMP, PULSE, SCALE, TSTAT

Product	Rev
AT6n00	1.0
AT6n50	n/a
610n	4.0
615n	n/a
620n	1.0
625n	n/a
6270	n/a

The Drive Resolution (DRES) command is used to match the indexer resolution to that of the motor/drive to which it is attached. This command is necessary in order to accurately calculate motor drive accelerations and velocities whether scaling is disabled (SCALEØ), or enabled. This command, in combination with the ERES command, provide the information required for encoder based moves.

Contouring Applications: All axes involved in contouring (those specified with the PAXES command) must have the same DRES setting.

Example:

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

DRESET Drive Reset

Type	Communication Interface
Syntax	<!>DRESET
Units	n/a
Range	n/a
Default	n/a
Response	n/a
See Also	RESET

Product	Rev
AT6n00	n/a
AT6n50	n/a
610n	n/a
615n	4.1
620n	n/a
625n	n/a
6270	n/a

DRESET *is applicable to the 6151 with 4.1 (or greater) software, and to the 6152/6154 with 4.2 (or greater) software.*

The DRESET command returns the internal motor drive to its power-on state and clears internal motor drive faults (see table below). The internal controller is not affected except that the command requires 2 seconds to complete. The DRESET command is an alternative to cycling AC power to the drive/controller electronics (**Control L1** and **Control L2** terminals) or activating the 615n's **Reset** input. Refer to the *615n Installation Guide* for further details on fault conditions and how to recover from them.

Note that you should rectify the cause of the fault before resetting the fault condition as noted in the table below; otherwise, the fault is likely to reoccur.

LED	Description	How to reset the fault
Bridge Fault *	Power stage over-temperature.	Note 1
	Power stage over-current.	Note 1
	Motor short circuit.	Note 1
Drive Fault *	Control board over-temperature.	Note 1
	Under-voltage (brownout).	Note 2
Motor Fault *	Resolver not connected.	Note 1
	Motor over-temperature.	Note 1
	Motor thermostat not connected.	Note 1
Over Voltage Fault *	Bus voltage exceeded 420VDC.	Note 1
Regen Fault *	Excessive regeneration (external regeneration resistor may be required).	Note 1

* When these faults occur, the 615n's output current is latched off.

Note 1 Activate the drive **RESET** input on the **DRIVE AUXILIARY** connector (hold the input to less than 1.0V for at least 20 milliseconds; reset begins upon release of the low voltage), issue the DRESET command, or cycle power to the **Control L1** and **Control L2** terminals.

Note 2 When the bus voltage drops below 85VAC 120VDC the **Drive Fault** LED will latch, indicating a under-voltage condition. The controller will then disable the drive. When the bus voltage has recovered there are 3 ways to clear the drive fault: (1) issue a reset via the drive reset input, (2) enter the DRESET command, or (3) enable the drive with the DRIVE1 command.

Reset Input or DRESET Command vs. RESET Command

There are two ways to reset the 615n:

- Reset the internal drive (2 options):
  - Issue the DRESET command.
  - Activate the **RESET** input, a *hardware* reset, to reset the drive functions within the 615n. (To reset the drive, hold the reset input at a low voltage, less than 1.0V, for at least 20 milliseconds. Reset will begin upon release of the low voltage.)
- Reset the internal controller functions with the RESET command. RESET is a *software* reset—it resets only the internal controller functions within the 615n.

The two reset functions (drive vs. controller) are independent, and do not directly affect each other. This means that if you use the **RESET** input or the DRESET command to reset the drive, the controller will remain up and running. Similarly, if you issue a RESET command to return the controller to power-up conditions, the drive functions will not automatically be reset.

DRFLVL Drive Fault Level

Type	Drive Configuration
Syntax	<!><@><a>DRFLVL<b><b><b><b>
Units	n/a
Range	b = 0 (active low), 1 (active high), or X (don't change)
Default	0 (1 for 6201 only)
Response	DRFLVL *DRFLVL0000 1DRFLVL *1DRFLVL0
See Also	[AS], [ASX], DRIVE, DRES, [ER], INFEN, TAS, TASX, TER

Product	Rev
AT6n00	1.0
OEM-AT6n00	n/a
AT6n50	1.0
610n	n/a
615n	n/a
6200	1.0
6201	n/a
625n	1.0
6270	1.0

The Drive Fault Level (DRFLVL) command is used to individually set the fault input level for each axis. To enable the drive fault input, the INFEN command must be enabled. Use the following table for setting the drive fault level for Compumotor drives.

Compumotor Product	Drive Fault Level
BLH, L, LE, PDS, PK130, PS7, UD2, UD5, UD12	Active Low (DRFLVLØ)
APEX, C+, Dynaserv, LN, OEM Series, S, SD (in SC rack), CD (in CN rack), TQ, Z, ZETA	Active High (DRFLVL1)

The drive fault input schematic is shown in the 6000 Series product *Installation Guide*.

Drive Fault Input Status:

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

Drive Fault Level (DRFLVL)	**Status of device driving the Fault input**	AS **bit #14 and** ASX **bit #4**
DRFLVL1 (active high)	OFF or not connected (not sinking current) 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 and program execution will be terminated. In **servo systems** (6250, etc.), motion is stopped at the rate set with the LHAD command (default is 100 units/sec/sec).

STEPPER SYSTEMS

A drive fault condition will stop motion instantaneously, without a controlled deceleration ramp—this allows the load to *free wheel*, possibly damaging equipment. Compumotor recommends using a brake on your motor drive system to brake the load in the event of a drive fault.

Example:

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

DRIVE Drive Enable

Type	Drive Configuration
Syntax	<!><@><a>DRIVE<b><b><b><b>
Units	n/a
Range	b = 0 (shutdown), 1 (enable), or X (don't change)
Default	1 (AT6n00, 610n & 620n); 0 (6250, AT6250 & 615n)
Response	DRIVE *DRIVE1111 1DRIVE *1DRIVE1
See Also	[AS], DRFLVL, DRES, KDRIVE, TER

Product	Rev
AT6n00	1.0
OEM-AT6n00	n/a
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Drive Enable command energizes (DRIVE1) or de-energizes (DRIVEØ) a Compumotor motor/drive combination. The internal shutdown output circuit is illustrated in the 6000 Series product's *Installation Guide*.

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

Servos: DRIVE1 energizes the motor drive (the SHTNO relay output is connected to COM, and the SHTNC relay output is disconnected from COM). DRIVEØ de-energizes the motor drive (the SHTNO relay output is disconnected from COM, and the SHTNC relay output is connected to COM). DRIVE1 also sets the commanded position (TPC) equal to the actual position (TPE). **NOTE**: If the Disable Drive on Kill (KDRIVE) mode is enabled, the drive will be de-energized in the event of a kill command or kill input.

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

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

DRPCHK RP240 Check

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

Product	Rev
AT6n00	n/a
AT6n50	n/a
610n	4.0
615n	4.1
620n	4.1
625n	4.1
6270	4.1

The Remote COM Port Check (DRPCHK) command is used to indicate whether a port is to be used with an RP240 or with 6000 language commands. The DRPCHK command affects the COM port selected with the last PORT command. The DRPCHK command value is automatically saved in battery-backed RAM.

NOTE: Depending on your product, COM1 is the RS-232 connector (or Rx, Tx, GND terminals on the AUX connector) or the COM1 connector; COM2 is the RP240 connector or the COM2 connector.

DRPCHKØ..... The serial port will be used for 6000 language commands. This is the default setting for COM1, and if using RS-485 half duplex on COM2. Power-up messages appear on all ports set to DRPCHKØ.

DRPCHK1..... A check for the presence of an RP240 will be performed at power-up/reset. If an RP240 is present, the 6000 product will initialize the RP240. If an RP240 is not present, the port may be used for 6000 language commands. Note that RP240 commands will be sent at power-up and reset.

DRPCHK2..... A status check for the presence of an RP240 will be periodically performed (every 5-6 seconds). If an RP240 is plugged in, the 6000 product will initialize the RP240. Press F6 on the RP240 periodically until the 6000 product recognizes the RP240. (The RP240 indicates that it has been recognized by beeping when F6 is pressed.)

DRPCHK3..... A status check for the presence of an RP240 will be performed at power-up/reset. If an RP240 is present, the 6000 product will initialize the RP240. If an RP240 is not present, no commands except DWRITE will have any effect for that port and the COM port will ignore received characters. This is the default setting for COM2, unless you are using RS-485 multi-drop communication (in which case the default changes to DRPCHKØ).

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

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

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

DVAR Display Variable on RP240

		Product	Rev
Type	Display (RP240) Interface	AT6n00	n/a
Syntax	<!>DVARi,<i>,<i>,<i>	AT6n50	n/a
Units	See below	610n	4.0
Range	n/a	615n	1.0
Default	See below	620n	1.0
Response	n/a	625n	1.0
See Also	[DREAD], [DREADF], DWRITE, VAR	6270	1.0

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

1st i = Variable number [Range 1 - 150]
2nd i = Number of whole digits displayed (left of decimal point) [Range 0 - 9]
3rd i = Number of fractional digits displayed (right of decimal point) [Range 0 - 8]
4th i = Sign bit: 0 = no sign displayed, 1 = display + or -

Example:

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

DWAVEF Waveform

		Product	Rev
Type	Drive Configuration	AT6n00	n/a
Syntax	<!><@><a>DWAVEF<i>	AT6n50	n/a
Units	n/a	610n	4.0
Range	1-4	615n	n/a
Default	1	620n	n/a
Response	DWAVEF *DWAVEF1 1DWAVEF *1DWAVEF1	625n	n/a
See Also	DAREN, DACTDP, DAUTOS, DMTIND, DMTSTT, DELVIS	6270	n/a

The Waveform (DWAVEF) command designates the waveform for the current from the 610n to the motor. The table below shows the amount of negative 3rd harmonic injection that corresponds to each of the four settings. In most application using Compumotor-supplied motors, the default setting provides the best performance.

<i>	3rd Harmonic Injection
1	-4%
2	-10%
3	-6%
4	Pure sine

Waveform Command Settings

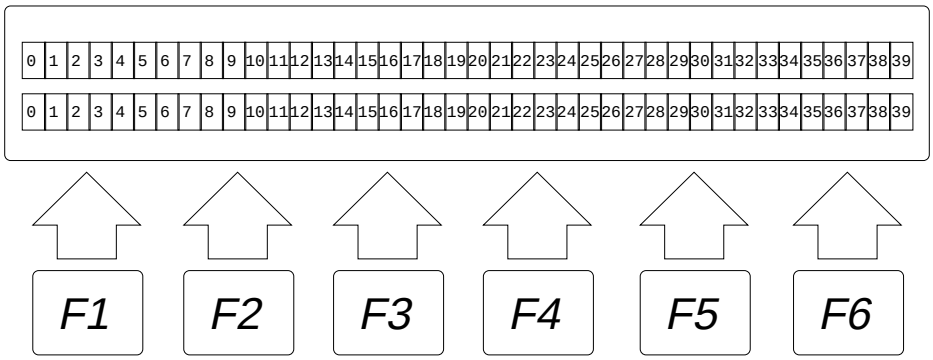
NOTE: If you choose the pure sine waveform when the motor resolution is set for 200 steps/rev, the Automatic Test mode of the 610n will be initiated. Refer to the *ZETA610n Installation Guide* for additional information about selecting a waveform.

DWRITE Write Text on RP240

		Product	Rev
Type	Display (RP240) Interface	AT6n00	n/a
Syntax	<!>DWRITE"message"	AT6n50	n/a
Units	n/a	610n	4.0
Range	Message can be ≤ 70 characters (may not use characters ", \, * or :)	615n	1.0
Default	See below	620n	1.0
Response	n/a	625n	1.0
See Also	DCLEAR, DLED, DPASS, DPCUR, DVAR, PORT	6270	1.0

The Write Text on RP240 (DWRITE) command displays a message on the RP240's LCD starting at the current cursor location. A message is a character string of up to 70 characters in length. The characters within the string may be any characters except quote ("), backslash (\), asterisk (*), and colon (:). Strings that have lower-case letters will be converted to upper case prior to display (see example).

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



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

Example:

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

E — Enable Communication

		Product	Rev
Type	Communication Interface	AT6n00	n/a
Syntax	<i_><!>E<b>	AT6n50	n/a
Units	i = unit number set by DIP switch or by ADDR command	610n	4.0

Range	i = 0 - 7 (if using DIP switch setting) or
	i = 0 - 99 (if unit # is set with the ADDR command);
	b = 0 (serial communication off) or 1 (serial communication on)
		615n	1.0

		620n	1.0
		625n	1.0
Default	b = 1	6270	1.0
Response	0_E: *E1		
See Also	ADDR, DRPCHK, ECHO, PORT, XONOFF		

The E command allows you to enable and disable serial ports on your stand-alone 6000 controller. To enable all units in the daisy-chain at one time, you can use the E1 command. The PORT command determines which COM port is affected by the E command.

Example:

```
PORT1          ; Next command affects the COM1 serial port on the 6000 product
0_E1           ; Enable serial port for unit with device address 0
```

ECHO — Communication Echo Enable

		Product	Rev
Type	Communication Interface	AT6n00	n/a
Syntax	<!>ECHO<b>	AT6n50	n/a
Units	n/a	610n	4.0
Range	b = 0 (disable), 1 (enable), 2 (echo through other COM port), 3 (echo through both COM ports), or X (don't change)	615n	1.0
		620n	1.0
Default	1	625n	1.0
Response	ECHO: *ECHO0	6270	1.0
See Also	], [, EOL, EOT, ERRLVL, PORT, [SS], TSS		

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

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

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

4.0 Enhancements: The ECHO command was enhanced with options 2 and 3. The purpose is to accommodate an RS-485 multi-drop configuration in which a host computer communicates to the "master" 6000 controller over RS-232 (COM1 port) and the master 6000 controller communicates over RS-485 (COM2 port) to the rest of the units on the multi-drop. For this configuration, the echo setup should be configured by sending to the master the following commands executed in the order shown. In this example, it is assumed that the master's device address is set to 1. Hence, each command is prefixed with "1_" to address only the master unit.

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

ELSE — Else Condition of IF Statement

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

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

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

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

Example:

```
IF(IN=b1X0)     ; Specify if condition to be input 1 = 1, input 3 = 0
T5              ; If condition evaluates true wait 5 seconds
ELSE            ; Else part of IF condition
TPE             ; If condition does not evaluate true transfer position of
                ; all encoders
NIF             ; End IF statement
```

EMOVDB — Encoder Move Deadband Enable

Type	Encoder Configuration
Syntax	<!><@><a>EMOVDB<b><b><b><b>
Units	n/a
Range	b = 0 (disable), 1 (enable), or X (don't change)
Default	0
Response	EMOVDB: *EMOVDB0000 1EMOVDB: *1EMOVDB0
See Also	ENC, EPM, EPMDB, EPMG, EPMV, ERES, ESDB, ESK, ESTALL

Product	Rev
AT6n00	1.0
OEM-AT6n00	n/a
AT6n50	n/a
610n	4.0
615n	n/a
620n	1.0
625n	n/a
6270	n/a

When this command is enabled on any axis, the next command in the command buffer will not be executed until the encoder on the enabled axis is within the position maintenance deadband (EPMDB).

NOTE

If EMOVDB is not enabled and position maintenance (EPM) is enabled, a move that requires end position adjustment (i.e. falls outside the EPMDB) will not correct its position if another move command is encountered. However, subsequent moves will not suffer from cumulative errors. This is due to the fact that the controller will command the motor to make the next move relative to where it should be and not where it actually is at the start of the move. Upon reaching its final destination, position maintenance will assure proper positioning of the motor.

Bit 19 of the axis status register reflects whether motion is within the deadband. The TAS command can be used to get the axis status response.

Example:

```
@EPMDB10        ; Position maintenance deadband set to 10 encoder steps on all axes
@EMOVDB1        ; Enable encoder move deadband on all axes
```

ENC — Encoder / Motor Step Mode

Type	Encoder Configuration
Syntax	<!><@><a>ENC<b><b><b><b>
Units	n/a
Range	b = 0 (motor step mode) or 1 (encoder step mode)
Default	0
Response	ENC: *ENC0000 1ENC: *1ENC0
See Also	[AS], EMOVDB, EPM, EPMDB, EPMG, EPMV, ERES, ESDB, ESK, ESTALL, FOLMAS, SCALE, SCLD, STRGTE, TAS, TPMAS

Product	Rev
AT6n00	1/0
OEM-AT6n00	n/a
AT6n50	n/a
610n	4.0
615n	n/a
620n	1.0
625n	n/a
6270	n/a

The Encoder/Motor Step Mode (ENC) command determines whether move distances are based on motor steps (ENCØ) or encoder steps (ENC1).

In motor step mode (ENCØ), the distance value (D) and the corresponding scaler (SCLD) are the only parameters used to determine actual step output values.

In encoder step mode (ENC1), the distance value (D) and the scaler (SCLD) are used to determine the number of encoder steps. This encoder step value is achieved by outputting as many motor steps as necessary to perform the required encoder steps.

NOTE
The acceleration, deceleration, and velocity parameters are always referenced in motor steps.

Enabling encoder step mode does not guarantee that a move will be positioned to the exact encoder step value commanded. Position maintenance (EPM) must be enabled to activate closed loop control.

Stalls can be detected by enabling Stall Detect (ESTALL), with an appropriate Stall Backlash Deadband (ESDB).

Example:

```
DRES25000,25000 ; Motor/drive resolution set to 25000 steps/rev on axes 1 and 2.
ERES4000,4000   ; Encoder resolution set to 4000 post-quadrature counts/rev
                ; on axes 1 & 2
SCALE1          ; Enable scaling
SCLA25000,25000 ; Set the acceleration scaling factor to 25000 steps/unit
                ; on axes 1 & 2
SCLV25000,25000 ; Set the velocity scaling factor to 25000 steps/unit
                ; on axes 1 & 2
SCLD1,1         ; Set the distance scaling factor to 1 step/unit on axes 1 & 2
ENC11XX         ; Encoder step mode for axes 1 & 2
MA00XX          ; Incremental index mode for axes 1 & 2
MC00XX          ; Preset index mode for axes 1 and 2
A10,12          ; Set the acceleration to 10 & 12 units/sec/sec for axes 1 & 2
V1,1            ; Set the velocity to 1 unit/sec for axes 1 & 2
D100000,1000    ; Set the distance to 100000 & 1000 units for axes 1 & 2
GO11            ; Initiate motion (axis 1 moves 100000 encoder steps,
                ; axis 2 moves 1000 encoder steps)
```

ENCPOL Encoder Polarity

Type	Encoder; Controller Configuration
Syntax	<!><@><a>ENCPOL<b><b><b><b>
Units	b = polarity bit
Range	0 (normal polarity), 1 (reverse polarity) or X (don't change)
Default	0
Response	ENCPOL *ENCPOL0000 1ENCPOL *1ENCPOL0
See Also	CMDDIR, [FB], FOLMAS, [PCE], [PE], [PER], PSET, SFB, TFB, TPE, TPCE, TPER

Product	Rev
AT6n00	4.0
OEM-AT6n00	n/a
AT6n50	n/a
610n	4.0
615n	n/a
620n	4.0
625n	n/a
6270	3.0

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

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

Immediately after issuing the ENCPOL command, the encoder will start counting in the opposite direction (including all encoder position registers). For servos, the polarity is immediately changed whether or not encoder feedback is currently selected with the SFB command (servos); for steppers, the polarity is immediately changed whether the controller in encoder step mode (ENC1) or motor step mode (ENCØ).

NOTE

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

The ENCPOL command is automatically saved in non-volatile RAM (stand-alone products only).

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

Example:

```
SFB1        ; Select encoder feedback for axis 1
SMPER100    ; Set maximum position error to 100 units on axis 1
PSETØ       ; Define current position of axis 1 as position zero
1TPE        ; *1TPE+0 (response indicates encoder #1 is at position zero)
MA0         ; Select incremental positioning mode
D+8000      ; Set distance to 8,000 units in the positive direction
GO1         ; Move axis 1. If the encoder polarity is incorrect, the axis will be
            ; unstable and will stop (drive disabled) as soon as the maximum
            ; position error of 100 units is reached.
1TPE        ; *1TPE-100 (response should show that encoder #1 is approximately at
            ; position -100; the minus sign indicates that the encoder is
            ; counting in the wrong direction)
ENCPOL1     ; Reverse encoder polarity on axis 1
PSET0       ; Define current position of axis 1 as position zero
DRIVE1      ; Enable the drive (drive was disabled when the SMPER value was
            ; exceeded)
D+8000      ; Set distance to 8,000 units in the positive direction
GO1         ; Move axis 1
1TPE        ; *1TPE+8000 (response shows encoder #1 has moved 8,000 units in the
            ; positive direction, indicating that the encoder is now counting in
            ; the correct direction)
```

END — End Program/Subroutine/Path Definition

		Product	Rev
Type	Program or Subroutine Definition	AT6n00	1.0
Syntax	<!>END	AT6n50	1.0
Units	n/a	610n	4.0
Range	n/a	615n	1.0
Default	n/a	620n	1.0
Response	n/a	625n	1.0
See Also	$, DEF, DEL, ERASE, GOBUF, GOSUB, GOTO, RUN	6270	1.0

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

Example:

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

EOL — End of Line Terminating Characters

		Product	Rev
Type	Communication Interface	AT6n00	1.0
Syntax	<!>EOL<i>,<i>,<i>	AT6n50	1.0
Units	n/a	610n	4.0
Range	i = 0 - 256	615n	1.0
Default	13,10,0	620n	1.0
Response	EOL: *EOL13,10,0	625n	1.0
See Also	], [, BOT, EOT, ERRLVL, PORT, WRITE, XONOFF	6270	1.0

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

Stand-alone products: The PORT command determines which COM port is affected by the EOL command.

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

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

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

Example:

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

EOT — End of Transmission Characters

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

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

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

Stand-alone products: The PORT command determines which COM port is affected by the EOT command.

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

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

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

Example:

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

EPM — Enable Position Maintenance Mode

Type	Encoder Configuration
Syntax	<!><@><a>EPM<b><b><b><b>
Units	n/a
Range	b = 0 (disable), 1 (enable), or X (don't change)
Default	0
Response	EPM: *EPM0000 1EPM: *1EPM0
See Also	[AS], ENC, EPMDB, EPMG, EPMV, TAS, TPER

Product	Rev
AT6n00	1.0
OEM-AT6n00	n/a
AT6n50	n/a
610n	4.0
615n	n/a
620n	1.0
625n	n/a
6270	n/a

This command enables position maintenance mode. This mode is only active while in encoder step mode (ENC1). When position maintenance mode is enabled, the actual end of move position is compared to the desired move position. If there is a difference between actual and desired position that is greater than the position maintenance deadband (EPMDB), a correction move will be generated to adjust for the discrepancy. The position error can be observed with the TPER command.

Do not mistake position maintenance for true servoing. Position maintenance is only invoked at the end of a move, where it continually monitors the position and corrects for position errors. Servoing takes place throughout the entire move, making adjustments on-the-fly.

If position maintenance is enabled and the motor drifts, check that the encoder is connected properly.

Example:

```
DRES25000,25000  ; Motor/drive resolution set to 25000 steps/rev on axes 1 and 2
ERES4000,4000    ; Encoder resolution set to 4000 post-quadrature counts/rev on
                 ; axes 1 and 2
SCALE1           ; Enable scaling
SCLA25000,25000  ; Set accel scaling factor to 25000 steps/unit on axes 1 & 2
SCLV25000,25000  ; Set velocity scaling factor to 25000 steps/unit on axes 1 & 2
SCLD1,1          ; Set the distance scaling factor to 1 step/unit on axes 1 & 2
EPMDB5,5         ; Position maintenance deadband set to 5 units on axes 1 & 2
ENC11XX          ; Encoder step mode for axes 1 and 2
EMOVDB11XX       ; Enable encoder move deadband
EPMG500,500      ; Set position maintenance gain factor to 500 Hz on axes 1 and 2
EPMV1,1          ; Set position maintenance correction velocity to 1 unit/sec or
                 ; 25000 steps/sec on axes 1 and 2
EPM11XX          ; Enable position maintenance on axes 1 and 2
MA00XX           ; Incremental index mode for axes 1 and 2
MC00XX           ; Preset index mode for axes 1 and 2
A10,12           ; Set the acceleration to 10 and 12 units/sec/sec for axes 1 & 2
V1,1             ; Set the velocity to 1 unit/sec for axes 1 and 2.
D100000,1000     ; Set the distance to 100000 and 1000 units for axes 1 and 2
GO11             ; Initiate motion on axes 1 and 2: axis 1 will move 100000
                 ; encoder steps and axis 2 will move 1000 encoder steps.
;
; (If, at the end of all the above moves, the actual encoder count is greater
; than 5 encoder steps away from the desired positions, a correction move will
; be made for each axis that exceeds the position maintenance deadband. The
; correction will be made at a maximum of 25000 steps/sec.)
```

EPMDB Position Maintenance Deadband

Type	Encoder Configuration
Syntax	<!><@><a>EPMDB<r>,<r>,<r>,<r>
Units	r = encoder counts
Range	0 - 99,999,999
Default	0
Response	EPMDB: *EPMDB0,0,0,0 1EPMDB: *1EPMDB0
See Also	[AS], ENC, EPM, EPMG, EPMV, TAS

Product	Rev
AT6n00	1.1
OEM-AT6n00	n/a
AT6n50	n/a
610n	4.0
615n	n/a
620n	1.0
625n	n/a
6270	n/a

The position maintenance deadband (EPMDB) command establishes the maximum encoder step error that is allowed at the end of a move. All EPMDB values entered are in encoder steps. At the end of a move, if position maintenance is enabled (EPM1), the difference between the actual encoder count and the desired encoder step move distance is continually monitored. Should the difference be greater than the position maintenance deadband, a correction move will occur.

This value also determines when the indexer considers itself *in position*. The status bit that reflects *in position* on a specific axis will not be set until the actual encoder count falls within the deadband.

Bit 19 of the axis status register reflects whether the axis is within the deadband. The TAS command can be used to get the axis status response.

Example: Refer to the enable position maintenance mode (EPM) command example.

EPMG — Position Maintenance Gain Factor

Type	Encoder Configuration
Syntax	<!><@><a>EPMG<i>,<i>,<i>,<i>
Units	i = gain value
Range	0 - 999,999
Default	10000
Response	EPMG: *EPMG10000,10000,10000,10000 1EPMG: *1EPMG10000
See Also	ENC, EPM, EPMDB, EPMV

Product	Rev
AT6n00	1.1
OEM-AT6n00	n/a
AT6n50	n/a
610n	4.0
615n	n/a
620n	1.2
625n	n/a
6270	n/a

The Position Maintenance Gain Factor (EPMG) command establishes the error correction velocity for the position maintenance error correction move. The correction velocity is determined by the EPMG value and the encoder error at the end of the move.

The correction velocity = EPMG * error.

The correction velocity will not exceed the maximum correction velocity value (EPMV).

Example: Refer to the enable position maintenance mode (EPM) command example.

EPMV — Position Maintenance Maximum Velocity

Type	Encoder Configuration
Syntax	<!><@><a>EPMV<r>,<r>,<r>,<r>
Units	r = units/sec
Range	0.00000 - 1,600,000 (Max. depends on PULSE setting)
Default	0.5000
Response	EPMV: *EPMV0.5000,0.5000,0.5000,0.5000 1EPMV: *1EPMV0.5000
See Also	ENC, EPM, EPMDB, EPMG, PULSE, SCALE, SCLV

Product	Rev
AT6n00	1.0
OEM-AT6n00	n/a
AT6n50	n/a
610n	4.0
615n	n/a
620n	1.0
625n	n/a
6270	n/a

The Position Maintenance Maximum Velocity (EPMV) command establishes the maximum velocity for any position maintenance correction move.

Velocities outside the valid range are flagged as an error, with a message *INVALID DATA-FIELD x, where x is the field number. When an invalid velocity is entered the previous velocity value is retained.

If scaling (SCALE) is **not** enabled, the velocity value is entered in motor revs/sec; this value is internally multiplied by the drive resolution (DRES) value to obtain a velocity value in motor steps/sec for motion calculations.

Scaling: If scaling (SCALE) is enabled, the EPMV command value is entered in user units/sec and is internally multiplied by the velocity scaling factor (SCLV) to convert user units/sec to motor steps/sec. The velocity value may be truncated if the value entered exceeds the velocity resolution at the given scaling factor. For further discussion on velocity scaling, refer to the SCLV command description.

Example: Refer to the enable position maintenance mode (EPM) command example.

[ER] Error Status

		Product	Rev
Type	Assignment or Comparison		
Syntax	See below	AT6n00	1.0
Units	n/a	AT6n50	1.0
Range	n/a	610n	4.0
Default	n/a	615n	1.0
Response	n/a	620n	1.0
See Also	[ASX], DRFLVL, ERROR, ERRORP, ESTALL, GOWHEN, INFNC, K, LDTUPD, LH, LS, S, SMPER, STRGTT, TASX, TER, TERF	625n	1.0
		6270	1.0

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

Syntax: VARBn=ER where n is the binary variable number, or [ER] can be used in an expression such as IF(ER=b11Ø1), or IF(ER=h7F).

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

The specific error-checking bits must be enabled by the Error-Checking Enable (ERROR) command before the ER command will provide an error response (see programming example below). The function of each axis status bit is shown below.

Bit #	**Function** (1 = Yes; Ø = No)
1*	Stall Detected: Functions when stall detection has been enabled (ESTALL). (n/a for OEM-AT6n00)
2	Hard Limit Hit: Functions when hard limits are enabled (LH).
3	Soft Limit Hit: Functions when soft limits are enabled (LS).
4	Drive Fault: Detected only of the drive is enabled (DRIVE) and the drive fault level is set correctly (DRFLVL and INFEN). (Drive Fault monitoring is n/a for OEM-AT6n00.) 610n: use ASX, TASX or TASXF to determine exact cause (ASX status is checked even if the drive is disabled).
5	RESERVED (refer to the ERROR command)
6	Kill Input: When an input is defined as a Kill input (INFNCi-C), and that input becomes active.
7	User Fault Input: When an input is defined as a User Fault input (INFNCi-F), and that input becomes active.
8	Stop Input: When an input is defined as a Stop input (INFNCi-D), and that input becomes active.
9	Stepper products—Pulse Cutoff (P-CUT): When the pulse cutoff input is activated (not grounded). Servo products—Enable input (ENBL): When the enable input is activated (not grounded). (n/a for OEM-AT6n00)
10	Pre-emptive (on-the-fly) GO or registration move profile not possible.
11**	Target Zone Settling Timeout Period (set with the STRGTT command) is exceeded.
12**	Maximum Position Error (set with the SMPER command) is exceeded.
13	RESERVED
14	Position relationship in GOWHEN already true when GO, GOL, FSHFC, or FSHFD was executed.
15***	LDT Position Read Error: Can be caused by LDT not connected, mechanical failure of LDT or LDTUPD command value too low.
16-32	RESERVED

* Stepper products only
** Servo products only
*** 6270 only

Example:

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

ERASE — Erase All Programs

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

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

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

ERES — Encoder Resolution

Type	Encoder Configuration
Syntax	<!><@><a>ERES<i>,<i>,<i>,<i>
Units	i = counts/rev
Range	1 - 65535 (steppers); 200 - 1024000 (servos)
Default	4000 (4096 for 615n only)
Response	ERES: *ERES4000,4000,4000,4000 1ERES: *1ERES4000
See Also	DRES, ENC, EPM, ESTALL, LDTRES, TSTAT

Product	Rev
AT6400	1.0
OEM-AT6n00	n/a
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

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

Servos: The servo system's resolution is determined by the resolution of the encoder used with the servo drive/motor system. The ERES command establishes the number of steps, or *counts* (post quadrature), per unit of travel. For example, Compumotor E Series encoders are 1,000-line encoders, and therefore have a 4,000 count/rev post-quadrature resolution (requires ERES4ØØØ). If the encoder is mounted directly to the motor, then to ensure that the motor will move according to the programmed distance and velocity, the controller's resolution (ERES value) must match the encoder's resolution. The 615n uses an internal 4096 count/rev resolver. *Remember that the 6270 can accept encoder feedback only on axis 1.*

Resolutions for Compumotor Encoders:

- E Series Encoders: ERES4000
- OEM Series Encoders (stepper): 83 size: ERES4000
57 Size: ERES2048
- SM Series Servo Motors: SMxxxxD-xxxx: ERES2000
SMxxxxE-xxxx: ERES4000
- OEM Series motors (servo): OEM2300E05A-MO: ERES2000
OEM2303E05A-MO: ERES2000
OEM3400E05A-MO: ERES2000
OEM3401E10A-MO: ERES2000
OEM2300E05A-MO: ERES4000
OEM2303E10A-MO: ERES4000
OEM3400E10A-MO: ERES4000
OEM3401E10A-MO: ERES4000
OEM2300E20A-MO: ERES8000
OEM2303E20A-MO: ERES8000
OEM3400E20A-MO: ERES8000
OEM3401E20A-MO: ERES8000

NOTE: When using the contouring feature, be sure that all axes involved are set at the same ERES value.

Example:
```
DRES25000,25000 ; Motor/drive resolution set to 25000 steps/rev on axes 1 & 2
                ; (for steppers only)
ERES4000,4000   ; Encoder resolution set to 4000 post-quadrature counts/rev on
                ; axes 1 & 2
SCALE1          ; Enable scaling
SCLA25000,25000 ; Set acceleration scaling factor to 25000 steps/unit/unit on
                ; axes 1 & 2
SCLV25000,25000 ; Set velocity scaling factor to 25000 steps/unit on axes 1 & 2
SCLD1,1         ; Set the distance scaling factor to 1 step/unit on axes 1 & 2
ENC11XX         ; Encoder step mode for axes 1 and 2 (for steppers only)
MA00XX          ; Incremental mode for axes 1 and 2
MC00XX          ; Preset mode for axes 1 and 2
A10,12          ; Set the acceleration to 10 and 12 units/sec/sec for axes 1 & 2
V1,1            ; Set the velocity to 1 unit/sec for axes 1 and 2
D100000,1000    ; Set the distance to 100000 and 1000 units for axes 1 and 2
GO11            ; Initiate motion on axes 1 and 2:
                ; Axis 1 will move 100000 encoder steps.
                ; Axis 2 will move 1000 encoder steps
```

ERRBAD Error Prompt

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

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

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

Stand-alone products: The PORT command determines which COM port is affected by the ERRBAD command.

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

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

ERRDEF Program Definition Prompt

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

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

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

Stand-alone products: The PORT command determines which COM port is affected by the ERRDEF command.

Example:

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

ERRLVL Error Detection Level

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

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Error Detection Level (ERRLVL) command specifies the format for all **Response** feedback and error messages (error messages are listed on page 7 of this manual, and in the Troubleshooting chapter of the *6000 Series Programmer's Guide*). Error level 4 is the default error detection level.

Stand-alone products: The PORT command determines which COM port is affected by the ERRLVL command.

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

ERROK Good Prompt

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

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Good Prompt (ERROK) command designates the characters to be placed into the output buffer after a command has been entered correctly. Up to 4 characters can be placed in the output buffer. These characters serve as a prompt for the next command. The characters are designated with their ASCII equivalent. For example, a carriage return is ASCII 13, a line feed is ASCII 10, a greater than symbol is ASCII 62, a space is ASCII 32, and no terminating character is designated with a zero. For a more complete list of ASCII equivalents, refer to the ASCII Table in Appendix C.

Stand-alone products: The PORT command determines which COM port is affected by the ERROK command.

Example:

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

ERROR Error-Checking Enable

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

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

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

The definition of each bit is as follows: *ERRORbbbb_bbbb_bbbb_bbbb_bbbb_bbbb_bbbb_bbbb

Bit #1 ^ Bit #32 ^

Bit #	**Function** (Error bits #8, #10, and #13 - #32 are reserved.)	**Branch Type**
1*	Stall Detected: Functions when stall detection has been enabled (ESTALL). ESK **must be enabled**. (n/a to OEM-AT6n00)	GOSUB
2	Hard Limit Hit: Functions when hard limits are enabled (LH).	GOTO if COMEXLØ; GOSUB if COMEXL1
3	Soft Limit Hit: Functions when soft limits are enabled (LS).	GOTO if COMEXLØ; GOSUB if COMEXL1
4	Drive Fault: The drive fault level must be set correctly (DRFLVL). (n/a to OEM-AT6n00) 610n: see ASX/TASXF for exact cause of fault.	GOTO
5	Commanded Kill or Commanded Stop: Whenever a !K, <ctrl>K, or !S is sent. **NOTE** If you want the program to stop, you must issue the !HALT command.	!K = GOTO; !S = GOTO if COMEXSØ; !S = GOSUB if COMEXS1, but need !C
6	Input Kill: When an input is defined as a KILL input (INFNCi-C), and that input becomes active.	GOTO

Bit #	Function (Error bits #8, #10, and #13 - #32 are reserved.)	Branch Type
7	User Fault Input: When an input is defined as a user fault input (INFNCi-F), and that input becomes active.	GOTO
8	Stop Input: When an input is defined as a user fault input (INFNCi-D), and that input becomes active.	GOTO
9	Stepper products—Pulse Cutoff (P-CUT): When pulse cutoff input is activated (not grounded). (n/a to OEM-AT6n00) Servo products—Enable input (ENBL): When enable input is activated (not grounded).	GOTO
10	Pre-emptive (on-the-fly) GO or registration move profile not possible at the time of attempted execution.	GOSUB
11**	Target Zone Settling Timeout Period (set with the STRGTT command) is exceeded.	GOSUB
12**	Maximum Position Error (set with the SMPER command) is exceeded.	GOSUB
14	GOWHEN condition already true when a subsequent GO, GOL, FSHFC, or FSHFD command is executed.	GOSUB
15	6270 Only – LDT Position Read Error	GOSUB

* Stepper products only; ** Servo products only
NOTE: Error bits 13 and 16-32 are reserved.

When error bit 5 (Commanded Kill or Stop) is enabled, a Stop (!S) or a Kill (!K or <ctrl>K) command will cause the controller to GOSUB or GOTO to the error program (ERRORP). The ERRORP program must be defined, and within the error program the cause of the error will need to be determined. The error status (ER) command can be used to determine the cause of the error. If none of the error status bits are set, the cause of the error is a commanded kill or a commanded stop. The reason for not setting a bit on this error condition is that there is no way to clear the error condition upon leaving the error program.

ERRORP Error Program Assignment

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

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

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

The purpose of the error program is to provide a programmed response to certain error conditions (see table below) that may occur during the operation of your system. Programmed responses typically include actions such as shutting down the drive(s), activating or de-activating outputs, etc. To detect and respond to the error conditions, the corresponding error-checking bit(s) must be enabled with the ERROR command (refer to the *ERROR Bit #* column in the table below). It is the programmer's responsibility to determine the cause of the error, and take action based on the error. The error condition can be determined using the ER evaluation in an IF statement (e.g., IF(ER=b1ØX)). An error program set-up example is provided in the *6000 Series Programmer's Guide*.

When an error condition occurs and the associated error-checking bit has been enabled with the ERROR command, the 6000 controller will branch to the error program. Depending on the error condition, the branch be either a GOTO or GOSUB. If the error condition calls for a GOSUB, then after the ERRORP

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

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

WHEN TO BRANCH

If you wish the branch to the error program to occur at the time the error condition is detected, use the continuous command execution mode (COMEXC1). Otherwise, the branch will not occur until motion on all axes has stopped.

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

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

NOTE

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

ERROR **Bit #**	**Cause of the Error**	**Branch Type to** ERRORP	**How to Remedy the Error**
1	Steppers Only: Stall detected (Stall Detection and Kill On Stall must be enabled first—see ESTALL and ESK, respectively) *n/a to OEM-AT6n00*	Gosub	Issue a GO command.
2	Hard Limit Hit (hard limits must be enabled first—see LH)	If COMEXLØ, then Goto; If COMEXL1, then Gosub	Change direction & issue GO command on the axis that hit the limit; or issue LHØ.
3	Soft Limit Hit (soft limits must be enabled first—see LS)	If COMEXLØ, then Goto; If COMEXL1, then Gosub	Change direction & issue GO command on the axis that hit the limit; or issue LSØ.
4	Drive Fault (Input Functions must be enabled—INFEN1; and Drive Fault Level must be correct—DRFLVL) *n/a to OEM-AT6n00*	Goto	Clear the fault condition at the drive, & issue a DRIVE1 command for the faulted axis.
5	Commanded Stop or Kill (whenever a !K, <ctrl>K, or !S command is sent)	If !K, then Goto; If !S & COMEXSØ, then Goto; If !S & COMEXS1, then Gosub, but need !C	No fault condition is present—there is no error to clear. If you want the program to stop, you must issue the !HALT command.
6	Kill Input Activated (see INFNCi-C)	Goto	Deactivate the kill input.

ERROR Bit #	Cause of the Error	Branch Type to ERRORP	How to Remedy the Error
7	User Fault Input Activated (see INFNCi-F)	Goto	Deactivate the user fault input, or disable it by assigning it a different function (INFNC).
8	Stop Input Activated (see INFNCi-D)	Goto	Deactivate the stop input, or disable it by assigning it a different function (INFNC).
9	Steppers: **P-CUT** input not grounded *n/a to OEM-AT6n00* Servos: **ENBL** input not grounded	Goto	Re-ground the **P-CUT** input (steppers) or the **ENBL** input (servos), and issue a DRIVE1111 command.
10	Pre-emptive (on-the-fly) GO or registration move profile not possible at the time of attempted execution.	Gosub	Issue another GO command.
11	Servos Only: Target Zone Timeout (STRGTT value has been exceeded)	Gosub	Issue these commands in this order: STRGTEØ, DØ, GO, STRGTE1
12	Servos Only: Exceeded Max. Allowable Position Error (set with the SMPER command).	Gosub	Issue a DRIVE1 command to the axis that exceeded the allowable position error. Verify that feedback device is working properly.
14	GOWHEN condition already true when GO, GOL, FSHFC, or FSHFD executed.	Goto	Issue another GOWHEN command.
15	Hydraulic Servos Only: LDT position read error due to bad connection, LDT failure, or LDTUPD value too small.	Gosub	Depending on cause, connect LDT, replace faulty LDT, or increase the LDTUPD value. Then issue DRIVE1 to the affected axis. To enable an axis without an LDT connected, connect **GATE+** to **GND**.

Reserved Bits: Bits 13 & 14, and 16 - 32 are reserved.

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

Example:

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

ESDB — Stall Backlash Deadband

Type	Encoder Configuration
Syntax	<!><@><a>ESDB<i>,<i>,<i>,<i>
Units	i = encoder steps
Range	0 - 99,999,999
Default	0
Response	ESDB: *ESDB0,0,0,0 1ESDB: *1ESDB0
See Also	[AS], DRES, ENC, ERES, ESK, ESTALL, TAS

Product	Rev
AT6n00	1.0
OEM-AT6n00	n/a
AT6n50	n/a
610n	4.0
15n	n/a
620n	1.0
625n	n/a
6270	n/a

The Stall Backlash Deadband (ESDB) command establishes the maximum number of encoder steps that a move can fall behind after a change in direction before stall detection is initiated. If there is no change in direction, the stall backlash deadband value will not be used to determine if there is a stall condition. A stall can be detected in either encoder step mode (ENC1) or motor step mode (ENCØ). To use the stall backlash deadband, stall detection (ESTALL) must be enabled.

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

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

ESK — Kill on Stall

Type	Encoder Configuration
Syntax	<!><@><a>ESK<b><b><b><b>
Units	n/a
Range	b = 0 (disable), 1 (enable), or X (don't change)
Default	0
Response	ESK: *ESK0000 1ESK: *1ESK0
See Also	DRES, ENC, ERES, ESDB, ESTALL

Product	Rev
AT6n00	1.0
OEM-AT6n00	n/a
AT6n50	n/a
610n	4.0
615n	n/a
620n	1.0
625n	n/a
6270	n/a

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

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

ESTALL — Enable Stall Detect

Type	Encoder Configuration
Syntax	<!><@><a>ESTALL<b><b><b><b>
Units	n/a
Range	b = 0 (disable), 1 (enable), or X (don't change)
Default	0
Response	ESTALL: *ESTALL0000 1ESTALL: *1ESTALL0
See Also	[AS], DRES, ENC, [ER], ERES, ESDB, ESK, TAS, TER

Product	Rev
AT6n00	1.4
OEM-AT6n00	n/a
AT6n50	n/a
610n	4.0
615n	n/a
620n	1.5
625n	n/a
6270	n/a

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

A stall condition will occur if the actual number of encoder counts received is less than expected for each motor step output segment. The number of encoder counts expected is determined by dividing the encoder resolution (ERES) by 100. The motor step output segment is determined by dividing the drive resolution (DRES) by 50. (Previous to revision 1.4 of the AT6n00 and revision 1.5 of the 6200, the stall detect algorithm would divide ERES by 50 and subtract 10, instead of dividing by 100.)

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

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

Stalls can be detected in either encoder step mode (ENC1) or motor step mode (ENCØ). To accurately detect a stall, the drive resolution (DRES) and the encoder resolution (ERES) must be properly set.

Example:

```
DRES25000,25000 ; Motor/drive resolution set to 25000 steps/rev on axes 1 and 2
ERES4000,4000   ; Encoder resolution set to 4000 post-quadrature counts/rev on
                ; axes 1 & 2
SCALE1          ; Enable scaling
SCLA25000,25000 ; Set the acceleration scaling factor to 25000 steps/unit/unit on
                ; axes 1 and 2
SCLV25000,25000 ; Set velocity scaling factor to 25000 steps/unit on axes 1 and 2
SCLD1,1         ; Set the distance scaling factor to 1 step/unit on axes 1 and 2
ESDB10,10       ; Stall backlash set to 10 motor steps on axes 1 and 2
ENC11XX         ; Encoder step mode for axes 1 and 2
ESTALL11XX      ; Enable stall detection on axes 1 and 2
ESK11XX         ; Enable kill on stall for axes 1 and 2
MA00XX          ; Incremental index mode for axes 1 and 2
MC00XX          ; Preset index mode for axes 1 and 2
A10,12          ; Set the acceleration to 10 and 12 units/sec/sec for axes 1 and 2
V1,1            ; Set the velocity to 1 unit/sec for axes 1 and 2
D100000,1000    ; Set the distance to 100000 and 1000 units for axes 1 and 2
GO11            ; Initiate motion on axes 1 and 2:
                ; Axis 1 will move 100000 encoder steps
                ; Axis 2 will move 1000 encoder steps
                ; (If, at any time during the above moves, any of the actual
                ; encoder counts fall behind a stall condition will be flagged,
                ; and motion will stop on the appropriate axis.)
```

FASTAT — Fast Status Customization

Type	Controller Configuration
Syntax	FASTAT<i>,<i>
Units	First integer (i) is the block number Second (i) is the configuration option number (1-23)
Range	First i = 3-8 for AT6n50, or 7-8 for AT6n00 Second i = 1-23
Default	n/a (no customization
Response	(see description below)
See Also	INDAX, SSFR, TANI, TANV, TAS, TDAC, TFB, TFS, [TIM], TIMST, TIN, TINO, TINT, TLIM, TNMCY, TOUT, TPANI, TPC, TPCA, TPCC, TPCE, TPER, TPMAS, TPSHF, TSS, TUS, TVEL, TVELA

Product	Rev
AT6n00	2.4
AT6n50	1.0
610n	n/a
615n	n/a
620n	n/a
625n	n/a
6270	n/a

Bus-based controllers have eight status area *blocks* of 8 words each. The default contents of each block are listed in the tables below (one table for servo products, one table for stepper products). You can use the FASTAT command to configure fast status area blocks to report other status information (you may customize blocks 3-8 on servo products, and blocks 7 & 8 on stepper products).

ABOUT THE FAST STATUS REGISTER

A fast status register is available to read various controller status data. The register occupies two bytes and is addressed two locations above the base address. For example, if the base address is at 300 Hex (768 decimal), the fast status register is at 302 Hex & 303 Hex (770 & 771 decimal).

Once a particular block is pointed to (by writing to Base+2), sequential reads from Base+2 and Base+3 will increment the pointer within the block. For instance, after reading the commanded position of axis 2, the next item read is the commanded position for axis 3. After reading the last item of a block, the pointer wraps around to the beginning of the block (i.e., axis 1 commanded/motor position).

TIP: The DOS support disk that ships with your product contains sample programs (see SAMPLES sub-directory) that access the data in the fast status registers.

Default Fast Status Register — SERVO PRODUCTS (AT6n50)

	HEX Offset in Fast Status Area	Description	Size	
Block 1	00	Commanded Position (counts), axis 1 – see TPC	2 words	
	02	Commanded Position (counts), axis 2	2 words	
	04	Commanded Position (counts), axis 3	2 words	
	06	Commanded Position (counts), axis 4	2 words	
Block 2	08	Feedback Device Position (counts), axis 1 – see TFB	2 words	
	0A	Feedback Device (counts), axis 2	2 words	
	0C	Feedback Device (counts), axis 3	2 words	
	0E	Feedback Device (counts), axis 4	2 words	
Block 3	10	Commanded Velocity (counts/sec), axis 1 – see TVEL	2 words	Box indicates customizable area of the fast status register. (Blocks 3–8)
	12	Commanded Velocity (counts/sec), axis 2	2 words	
	14	Commanded Velocity (counts/sec), axis 3	2 words	
	16	Commanded Velocity (counts/sec), axis 4	2 words	
Block 4	18	Axis Status Information, axis 1 – see TAS	2 words	
	1A	Axis Status Information, axis 2	2 words	
	1C	Axis Status Information, axis 3	2 words	
	1E	Axis Status Information, axis 4	2 words	
Block 5	20	Input Status for 28 inputs (bits 0-27) – see TIN	2 words	
	22	Output Status for 28 outputs (bits 0-27) – see TOUT	2 words	
	24	Limits – see *Limits Bit Assignments* table below	2 words	
	25	Other Input Status – see TINO	2 words	
	26	Analog Voltage, channel 4,3,2,1 – see TANV	2 words	
Block 6	28	Interrupt Status – see TINT	2 words	
	2A	System Status – see TSS	2 words	
	2C	User Status – see TUS	1 word	
	2D	Time Frame Mark (system update units) – see SSFR	1 word	
	2E	Programmable Timer Value – see TIMST	2 words	
Block 7	30	ANI input value, input 1 (ADC counts) – see TPANI	1 word	819 ADC counts/volt
	31	ANI input value, input 2 (ADC counts)	1 word	
	32	ANI input value, input 3 (ADC counts)	1 word	
	33	ANI input value, input 4 (ADC counts)	1 word	
	34	Commanded DAC value, axis 1 (DAC counts) – see TDAC	1 word	2048 DAC counts/10 volts
	35	Commanded DAC value, axis 2 (DAC counts)	1 word	
	36	Commanded DAC value, axis 3 (DAC counts)	1 word	
	37	Commanded DAC value, axis 4 (DAC counts)	1 word	
Block 8	38	Position error, axis 1 (counts) – see TPER	2 words	
	3A	Position error, axis 2 (counts)	2 words	
	3C	Position error, axis 3 (counts)	2 words	
	3E	Position error, axis 4 (counts)	2 words	

Limits Bit Assignments (from block 5):

0 = Axis 1 positive hardware end-of-travel limit
1 = Axis 1 negative hardware end-of-travel limit
2 = Axis 2 positive hardware end-of-travel limit
3 = Axis 2 negative hardware end-of-travel limit
4 = Axis 3 positive hardware end-of-travel limit
5 = Axis 3 negative hardware end-of-travel limit
6 = Axis 4 positive hardware end-of-travel limit
7 = Axis 4 negative hardware end-of-travel limit
8 = Axis 1 home limit
9 = Axis 2 home limit
10 = Axis 3 home limit
11 = Axis 4 home limit

***NOTE**: These bits report the current state of the input, not necessarily whether a limit has been encountered.*

Default Fast Status Register — STEPPER PRODUCTS (AT6n00)

	HEX Offset in Fast Status Area	Description		Size
Block 1	00	Motor Position (steps)	Axis 1 – see TPM	2 words
	02		Axis 2	2 words
	04		Axis 3	2 words
	06		Axis 4	2 words
Block 2	08	Encoder Position (counts)	Axis 1 – see TPE	2 words
	0A		Axis 2	2 words
	0C		Axis 3	2 words
	0E		Axis 4	2 words
Block 3	10	Velocity (steps/sec)	Axis 1 – see TVEL	2 words
	12		Axis 2	2 words
	14		Axis 3	2 words
	16		Axis 4	2 words
Block 4	18	Axis Status Information	Axis 1 – see TAS	2 words
	1A		Axis 2	2 words
	1C		Axis 3	2 words
	1E		Axis 4	2 words
Block 5	20	Input Status for 28 inputs (bits 0-27) – see TIN		2 words
	22	Output Status for 24 outputs (bits 0-23) – see TOUT		2 words
	24	Limits – see *Limits Bit Assignments* table above		2 words
	25	Other Input Status – see TINO		2 words
	26	Analog Voltage, channel 4,3,2,1 – see TANV		2 words
Block 6	28	Interrupt Status – see TINT		2 words
	2A	System Status – see TSS		2 words
	2C	User Status – see TUS		2 words
	2D	Time Frame Mark (2ms timer)		2 words
	2E	Programmable Timer Value – see TIMST		2 words
Block 7	30	Motor position captured with trigger A, axis 1 – see TPCMA		2 words
	32	Motor position captured with trigger A, axis 2		2 words
	34	Motor position captured with trigger A, axis 3		2 words
	36	Motor position captured with trigger A, axis 4		2 words
Block 8	38	Motor position captured with trigger B, axis 1 – see TPCMB		2 words
	3A	Motor position captured with trigger B, axis 2		2 words
	3C	Motor position captured with trigger B, axis 3		2 words
	3E	Motor position captured with trigger B, axis 4		2 words

Box indicates customizable area of the fast status register.

Customizing the Fast Status Register

The number of customizable blocks differs between servo (AT6n50) and a stepper (AT6n00) products:

AT6n50: You can customize status blocks 3-8 (blocks 1 & 2 may not be changed).

AT6n00: You can customize status blocks 7 & 8 (blocks 1-6 may not be changed).

The FASTAT command syntax is FASTAT<i>,<i>. In the first data field ("<i>"), enter the number of the block you wish to change. In the second data field, enter the number of the content option (see table below). For example, the FASTAT5,6 command configures block #5 to report the position error (content option #6).

To check the current configuration of all blocks in the status area, type "FASTAT" followed by a carriage return; the controller will respond with the current option number selected for each block. To check the configuration of one block, type "FASTATi" (i = number of block in question) followed by a carriage return.

It takes one system update period to process the FASTAT command and re-configure the fast status blocks. If using the AT6n50 refer to the table in the SSFR command description to determine the system update period (affected by the SSFR and INDAX command settings). For the AT6n00, one system update period is 2ms.

Option #	Information Provided *	Equivalent to	Size (words)	AT6n50	AT6n00
1	Commanded velocity (counts/sec)	TVEL	2 x 4	x	x
2	Axis status	TAS	2 x 1	x	x
3	Programmable input status (including triggers)	TIN	2 x 1	x	x
	Programmable output status (including aux. outputs)	TOUT	2 x 1	x	x
	Limit status (hardware end-of-travel and home)	**	1 x 1	x	x
	Other input status (joystick and enable)	TINO	1 x 1	x	x
	A/D analog input voltage (joystick connector)	TANV	2 x 1	x	x
4	Interrupt status	TINT	2 x 1	x	x
	System status	TSS	2 x 1	x	x
	User status	TUS	1 x 1	x	x
	Time frame-mark (system update units – see SSFR)	n/a	1 x 1	x	x
	Time value (milliseconds)	TIM	2 x 1	x	x
5	ANI input counts (ANI option only) (819 ADC counts/volt)	TANI	1 x 4	x	
	Commanded DAC counts (2048 DAC counts/10 volts)	TDAC	1 x 4	x	
6	Position error (counts)	TPER	2 x 4	x	x
7	Following status	TFS	2 x 4	x	***
8	Actual velocity (feedback device counts/sec)	TVELA	2 x 4	x	
9	Captured commanded/motor position via trigger A (counts)	TPCCA/TPCMA	2 x 4	x	x
10	Captured commanded/motor position via trigger B (counts)	TPCCB/TPCMB	2 x 4	x	x
11	Captured commanded/motor position via trigger C (counts)	TPCCC/TPCMC	2 x 4	x	x
12	Captured commanded/motor position via trigger D (counts)	TPCCD/TPCMD	2 x 4	x	x
13	Captured actual position via trigger A (counts)	TPCEA	2 x 4	x	x
14	Captured actual position via trigger B (counts)	TPCEB	2 x 4	x	x
15	Captured actual position via trigger C (counts)	TPCEC	2 x 4	x	x
16	Captured actual position via trigger D (counts)	TPCED	2 x 4	x	x
17	Captured ANI value via trigger A (ADC counts)	TPCAA	2 x 4	x	
18	Captured ANI value via trigger B (ADC counts)	TPCAB	2 x 4	x	
19	Captured ANI value via trigger C (ADC counts)	TPCAC	2 x 4	x	
20	Captured ANI value via trigger D (ADC counts)	TPCAD	2 x 4	x	
21	Following master cycle position	TPMAS	2 x 4	x	***
22	Following master cycle number	TNMCY	2 x 4	x	***
23	Following net shift	TPSHF	2 x 4	x	***

* Motion data in the fast status area is never scaled.
Any data that is not applicable (e.g., 3rd and 4th axis information for AT6250 & AT6200) will be zeros.
** Refer to the *Limits Bit Assignments* table above.
*** Available for stepper product revisions 3.0 and higher.

Example:

```
FASTAT4,7       ; Configure fast status block #4 to contain the status for
                ; Following
```

[FB] Value of Current Feedback Device

Type	Servo; Assignment or Comparison
Syntax	See below
Units	See below
Range	See below
Default	n/a
Response	n/a
See Also	[ANI], ANIPOL, CMDDIR, ENCPOL, GOWHEN, [LDT], LDTPOL, [PANI], [PE], PSET, SCALE, SCLD, SFB, TFB, TPANI

Product	Rev
AT6n00	n/a
AT6n50	1.0
610n	n/a
615n	1.0
620n	n/a
625n	3.0
6270	1.0

Use the FB command to assign the value of one of the current feedback devices to a variable or to make a comparison. Depending on the configuration of the SFB command, the feedback device could be an encoder or an ANI analog input (-ANI option only), or an LDT if using the 6270, or an internal resolver if using the 615n. *The 6270 cannot accept encoder feedback on axis 2.*

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

If scaling is **not** enabled, the position values returned will be encoder, LDT, resolver or ANI counts. If scaling is enabled (SCALE1), the encoder, LDT, resolver and ANI values will be scaled by the SCLD value.

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

Example:
```
SFB1,3          ; Feedback for axis is encoder #1; feedback for axis two is LDT #2
VAR6=2FB        ; Assign position (scalable) of LDT #2 (axis 2) to variable #6
IF(1FB<500)     ; If position (scalable) of encoder #1 (axis 1) is less than 500,
                ; do the commands following the IF statement until the NIF command
VAR4=1FB+1000   ; Set variable #4 equal to current position of encoder plus 1,000
NIF             ; End of IF statement
```

FFILT — Following Filter

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

Product	Rev
AT6n00	3.0
AT6n50	3.0
610n	4.0
615n	3.0
620n	3.0
625n	3.0
6270	3.0

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

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

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

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

Increasing the FFILT command value increases the filtering effect (lowers the bandwidth), but also increases the phase tracking error of the slave axis, resulting in a phase lag. Therefore, if an application requires high slave tracking accuracy, do not use heavy filtering. However, if slave tracking error is not critical, then you can increase master filtering to help alleviate sensor noise or master vibration problems.

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

FMAXA Slave Maximum Acceleration

		Product	Rev
Type	Following	AT6n00	3.0
Syntax	<!><@><a>FMAXA<r>,<r>,<r>,<r>	AT6n50	n/a
Units	r = units/sec/sec	610n	4.0
Range	r = 0.00025 - 24999999.9999 (scalable with SCLA)	615n	n/a
Default	0.00 (no limit imposed)	620n	3.0
Response	FMAXA *FMAXA0.00,0.00,0.00,0.00 1FMAXA *FMAXA0.00	625n	n/a
See Also	FFILT, FMAXV, FPPEN, SCLA	6270	n/a

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

As part of a ramp to new ratio, or simply following an accelerating master at constant ratio, a slave may be required to accelerate. If the required acceleration is larger than FMAXA, the slave will begin falling behind its commanded position. The 6000 controller will attempt to make up this position error as soon as the commanded accel falls below FMAXA. In stepper controllers, an error correction velocity is added to that implied by the commanded ratio. The velocity used to make up the error is limited to that specified with EPMV.

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

Example:

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

FMAXV Slave Maximum Velocity

		Product	Rev
Type	Following	AT6n00	3.0
Syntax	<!><@><a>FMAXV<r>,<r>,<r>,<r>	AT6n50	n/a
Units	r = units/sec (scalable with SCLV)	610n	4.0
Range	r = 0.000000 - 1600000.000000	615n	n/a
Default	0.00 (no limit imposed)	620n	3.0
Response	FMAXV *FMAXV0.00,0.00,0.00,0.00 1FMAXV *FMAXV0.00	625n	n/a
See Also	FFILT, FMAXA, FPPEN, SCLV	6270	n/a

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

Normally in a Following application, the slave velocities will be known based on the normal speed of the master and the commanded Following ratios (FOLRN and FOLRD). In some cases, however, the master speed may be higher than normal, the slave may be commanded to perform a shift move, or some other event may occur which will cause the slave to travel at a velocity higher than expected. In these cases, the 6000 controller will increase the speed of the slave as necessary to perform the required move, but only up to the FMAXV value.

If the commanded speed is higher than FMAXV, the slave axis will start falling behind its commanded position. The 6000 controller will attempt to make up this position error as soon as the commanded speed falls below FMAXV. In stepper controllers, an error correction velocity is automatically added to that implied by the commanded ratio. The velocity used to overcome the error is limited to that specified with EPMV.

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

Example:

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

FMCLEN Master Cycle Length

		Product	Rev
Type	Following	AT6n00	3.0
Syntax	<!><@><a>FMCLEN<r>,<r>,<r>,<r>	AT6n50	3.0
Units	r = master distance units (scalable)	610n	4.0
Range	r = 0 - 999,999,999 (scalable with SCLMAS)	615n	3.0
Default	0	620n	3.0
Response	FMCLEN *FMCLEN0,0,0,0 1FMCLEN *1FMCLEN0	625n	3.0
See Also	FMCNEW, FMCP, FOLEN, [FS], GOWHEN, [PMAS], SCLMAS, TFS, TPMAS, WAIT	6270	3.0

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

The concept of a master cycle may be useful when moves or other events must be initiated at certain master positions in a repetitive cycle. By specifying a master cycle length, periodic actions may be programmed in a loop or with subroutines which refer to cycle positions, even if the master runs continuously. It is possible to program the 6000 controller to suspend program operation or delay moves until specified master cycle positions. The master cycle length, FMCLEN, should be defined before the functions which wait for periodic master cycle positions are used. An axis need not be in Following mode (FOLEN1) to utilize the concept of a master cycle. However, **master positions will not be measured until a master has been assigned with the** FOLMAS **command.**

Example (refer also to FOLEN example #2)**:**

```
SCLMAS4000,16000  ; Set the master scale factors: axis 1 = 4000; axis 2 = 16000
FMCLEN3,(VAR2)    ; Set axis 1 master cycle length to 3 user units, and axis 2
                  ; to the value of variable 2 times the SCLMAS value
```

FMCNEW Restart Master Cycle Counting

		Product	Rev
Type	Following	AT6n00	3.0
Syntax	<!><@><a>FMCNEW<b><b><b><b>	AT6n50	3.0
Units	n/a	610n	4.0
Range	b = 0 (do not restart), 1 (restart immediately), or X (don't change)	615n	3.0
Default	n/a	620n	3.0
Response	n/a	625n	3.0
See Also	FMCLEN, FMCP, GOWHEN, [NMCY], [PMAS], TPMAS, TRGFN, WAIT	6270	3.0

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

The function of the FMCNEW1 command can be initiated with a trigger input by specifying a TRGFNcx1 command. If the FMCNEW1 command is used, master cycle counting is restarted immediately, if TRGFNcx1 is used, the 6000 controller will record the instruction to set the master cycle position when the specified trigger occurs. In this case, the master cycle counting is restarted when the specified trigger is activated, even though commands continue to execute and the master cycle counting continues.

FMCNEWØ or FMCNEW1 will remove the status of master cycle restart pending a trigger input (TRGFNcx1). In the case of FMCNEWØ, no restart will occur, and the specified trigger will not cause a new cycle restart. Furthermore, if there is a trigger-based restart pending on axis X, and on axis Y a GOWHEN condition is specified based on PMAS of axis X, then issuing an FMCNEWØ on axis X will clear the pending trigger on axis X and will also clear the pending GOWHEN on axis Y.

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

Example:

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

FMCP — Initial Master Cycle Position

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

Product	Rev
AT6n00	3.0
AT6n50	3.0
610n	4.0
615n	3.0
620n	3.0
625n	3.0
6270	3.0

The FMCP command defines the initial master cycle position in user units. The initial master cycle position is assigned as the current master cycle position each time master cycle counting is restarted with the FMCNEW or TRGFNcx1 command. This value is scaled by the SCLMAS parameter. Numeric variables (VAR) can be used with this command. The default value for FMCP is zero (FMCPØ), which means that the master cycle position will be zero when master cycle counting is restarted (see FMCNEW).

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

For example, suppose FMCLEN was set to 20 and FMCP was set to 7. When master cycle position counting is restarted, either via FMCNEW1 or the specified trigger (TRGFNcx1), the initial master cycle position will be 7. Rollover will occur after the master travels 13 more units, and the master cycle position would go to zero.

Example:

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

FOLEN — Following Mode Enable

Type	Following
Syntax	<!><@><a>FOLEN<b><b><b><b>
Units	n/a
Range	b = 0 (disable), 1 (enable) or X (don't change)
Default	0
Response	FOLEN: *FOLEN0000 1FOLEN: *1FOLEN0
See Also	FOLK, FOLMAS, FOLRD, FOLRN, FOLRNF, FOLSND, [FS], FSHFC, FSHFD, GOWHEN, JOG, JOY, TFS

Product	Rev
AT6n00	3.0
AT6n50	3.0
610n	4.0
615n	3.0
620n	3.0
625n	3.0
6270	3.0

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

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

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

Servo Users: S-Curve profiling is not operational during Following moves.

RESTRICTIONS ON USING FOLEN

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

- You may not enable Following (FOLEN1) on an axis that is in motion, waiting for a GOWHEN condition, or operating in the Joystick mode (JOY1) or Jog mode (JOG1).
- You may not disable Following (FOLEN0) on an axis that is in motion (unless moving at ratio in continuous mode, MC1, and not shifting) or waiting for a GOWHEN condition.

FOLEN **Examples**

Example #1:

The 6000 product is controlling a rotary drive, the master is a 1000-line incremental encoder mounted on the back of an externally controlled motor, and programming units are to be revs/second (rps).

Stepper Products:

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

Scaling Set Up: (prior to defining program)

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

Servo Products:

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

Scaling Set Up: (prior to defining program)

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

The application program is defined as follows:

```
DEL FOLTST    ; Delete program called FOLTST
DEF FOLTST    ; Begin definition of program called FOLTST
INFEN1        ; Enable input functions
INFNC25-H     ; Set input #25 (TRG-A) to be "trigger interrupt" (used with GOWHEN later)
COMEXC1       ; Select continuous command processing mode
MC1           ; Select continuous positioning mode
FOLMAS31      ; Assign encoder input #3 as master for axis #1
FOLMD1        ; Slave should change ratios over 1 master revolution
FMCLEN100     ; Set master cycle length to 100 revs
FOLRD1        ; Set slave-to-master Following ratio denominator to 1
              ; (applies to all subsequent FOLRN commands)
FOLEN1        ; Enable Following on axis #1
D+            ; Set motion to the positive- direction
$STRMV        ; Label to repeat move
TRGFNA1       ; Suspend execution of next move until trigger (TRG-A) is active
TRGFNAx1      ; Begin new master cycle (counter at 0) when trigger (TRG-A) is active
FOLRN1        ; Set slave-to-master Following ratio numerator to 1 (ratio set to 1:1)
GO1           ; Start continuous Following move (when TRG-A is active)
WAIT(1AS.26=b0 AND FS.4=b1) ; Wait for profile to actually start
                            ; (when TRG- A is active) and be at ratio
GOWHEN(1PMAS>=25)  ; Suspend execution of next move until master position >= 25
FOLRN0.5      ; Set Following ratio numerator to 0.5 (ratio set to 0.5:1)
GO1           ; Initiate new move according to new Following ratio
              ; (when master position >= 25)
WAIT(1AS.26=b0 AND FS.4=b1)  ; Wait for profile to actually start
                             ; (when master position >= 25) and be at ratio
GOWHEN(1PMAS>=75)  ; Suspend execution of next move until master position >= 75
FOLRN0        ; Set Following ratio numerator to zero
              ; (ratio causes slave to ramp to stop)
GO1           ; Initiate new move with new Following ratio (when master pos. >= 75)
WAIT(1AS.26=b0 AND FS.1=b0)  ; Wait for profile to actually start
              ; (when master position >= 75) and the slave is not moving
JUMP STRMV    ; Repeat the cycle
END           ; End of program
```

Example #2:

Stepper Products:

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

Scaling Set Up: (prior to defining program)

```
SCALE1          ; Enable scaling
SCLA100000      ; Set accel scaling: 100,000 steps/inch
SCLV100000      ; Set velocity scaling: 100,000 steps/inch
SCLD100000      ; Set slave distance scaling: 100,000 steps/inch
SCLMAS1600      ; Set master scale factor to 16000 steps/inch to program in inches
```

Servo Products:

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

Scaling Set Up: (prior to defining program)

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

The application program is defined as follows:

```
DEF STAMPR          ; Begin definition of program called STAMPR
COMEXS1             ; Continue command execution after Stop
COMEXC1             ; Continue command execution during motion
SCALE1              ; Enable parameter scaling
OUTFEN1             ; Enable programmable output functions
OUTFNC7-A           ; Configure output #7 as a general-purpose prog. output
INFEN1              ; Enable programmable input functions
INFNC26-H           ; Define TRG-B as trigger interrupt (use as GOWHEN input)
A10                 ; Acceleration = 10 inches/sec/sec
V5                  ; Velocity = 5 inches/sec (non-Following moves)
MA1                 ; Enable absolute positioning mode for axis 1
FOLMAS21            ; Assign encoder input #2 as master for axis 1
FOLRN1              ; Set slave-to-master Following ratio numerator to 1
FOLRD1              ; Set slave-to-master Following ratio denominator to 1 (ratio is 1:1)
FOLMD1              ; Accel the slave over 1 master inch for Following moves
FMCLEN40            ; Master cycle length is 40 inches
$INKON              ; Label to repeat inking process
FOLEN1              ; Enable Following on axis #1
TRGFNBx1            ; Begin new master cycle when TRG-B goes active
                    ; (product sensed on conveyor)
TRGFNB1             ; Start next move when trigger #2 is active
D+                  ; Set to positive-direction
MC1                 ; Select continuous positioning mode
GO1                 ; Start continuous slave move on trigger #2
WAIT(1PMAS>=10.5)   ; Wait until master position is 10.5 inches - this is when the
                    ; stamping device can be actuated without mechanical damage
                    ; to the leadscrew assembly
OUT.7-1             ; Turn on actuator (output #7) to place ink stamp on product
T.1                 ; Wait for the ink stamp to be pressed in place by a
                    ; stationary stamper
OUT.7-0             ; Turn off actuator (output #7)
S1                  ; Stop slave move
WAIT(1AS.1=b0)      ; Wait until the axis is not moving
FOLEN0              ; Disable Following on axis #1
D0                  ; Set distance (position) to zero
MC0                 ; Select preset positioning mode
GO1                 ; Move back to zero (the home position)
WAIT(MOV=b0)        ; Wait until the axis is not moving
JUMP INKON          ; Begin cycle again on trigger #2
END                 ; End of program
```

FOLK — Following Kill

Type	Following
Syntax	<!>FOLK<b><b><b><b>
Units	n/a
Range	b= 0 (disable) or 1 (enable)
Default	0
Response	FOLK *FOLK0000
See Also	DRIVE, [ER], ERROR, FOLEN, FOLRD, FOLRN, FOLMAS, FOLMD, FSHFC, FSHFD, INFNC, K, [PSHF], SMPER, TER

Product	Rev
AT6n00	n/a
AT6n50	4.5
610n	n/a
615n	4.5
620n	n/a
625n	4.5
6270	4.5

Under default operation (FOLK0), certain error conditions (i.e., drive fault input active, or max. position error limit exceeded) will cause the 6000 controller to disable the drive (i.e., remove torque) and to kill the Following profile (slave's commanded position loses synchronization with the master).

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

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

FOLMAS — Assignment of Master to Slave

Type	Following
Syntax	<!><@><a>FOLMAS<±ii>,<±ii>,<±ii>,<±ii>
Units	1st i = master axis #; 2nd i = master count source; ± sets direction of master counts relative to direction of actual master count source
Range	1st i=1-4 (axis); 2nd i=1 (encoder), 2 (ANI input), 3 (LDT), or 4 (commanded position)
Default	+0 (disable from being a Following slave)
Response	FOLMAS *FOLMAS+0,+0,+0,+0 1FOLMAS *1FOLMAS+0
See Also	ENC, FOLEN, FOLK, FOLMD, FOLRD, FOLRN, FOLRNF, [FS], TFS

Product	Rev
AT6n00	3.0
AT6n50	3.0
610n	4.0
615n	3.0
620n	3.0
625n	3.0
6270	3.0

FOLMAS is used to assign or un-assign a master to a slave.

Each data field (±ii) configures that axis as a slave following the specified master count source. In the syntax for each slave axis (±ii), the sign bit sets the direction of master counting relative to the actual direction of the counts as received from the master count source. The first i selects the axis number of the master you are assigning to the slave, and the second i selects the count source of that master axis. **If a zero (ø) is placed in the data field (±ii), that axis becomes a normal non-Following axis.**

The availability of master count sources differs by product, as indicated in the table below.

Product	Options for Second i	Measurement *
AT6n00, 620n, & OEM6200	1—Encoder 4—Commanded (motor) position	Encoder counts Motor counts
AT6n50, 625n & OEM6250	1—Encoder 2—ANI input **(ANI option only)** 4—Commanded position	Encoder counts ADC counts Feedback device counts
610n	1—Encoder	Encoder counts
615n	1—Encoder 2—ANI input **(ANI option only)**	Encoder counts ADC counts
6270	1—Encoder (axis 1 only) 2—ANI input (6270-ANI only) 3—LDT 4—Commanded position	Encoder counts ADC counts LDT counts Feedback device counts

***** If scaling is enabled (SCALE1), the measurement of the master is scaled by the SCLMAS value.

NOTE

- A slave axis cannot use its own feedback device or commanded position as the master input.
- Multiple axes may be slaved to the same count source (e.g., encoder) from the same master. However, multiple axes may **not** be slaved to different count sources (e.g., encoder and commanded position) from the same master.

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

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

NOTE

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

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

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

There are several applications in which a minus sign in the `FOLMAS` command is used. A minus sign should be used whenever the master is moving in the desired positive direction and yet the 6000 controller actually perceives the master to be moving in the negative direction. For example, this can occur when the master input device is mounted on the opposite side of a conveyor. Putting a minus sign in front of the master parameter specification in the `FOLMAS` command causes the incoming master signal to be negated before it is used by the slave. The term *master count* refers to the count after negation, if any.

For preset slave moves, the direction the slave travels depends on the mode of operation (absolute or incremental) and the commanded position. However, once a preset slave move is commanded, it will only start moving if the master is moving in the positive direction. This is true no matter the commanded direction of the slave move.

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

Example: (refer also to the `FOLEN` examples)

```
FOLMAS31,31     : Define axis 1 and 2 to be slaves to the encoder signal received
                ; on axis 3. Axis 3 encoder is the master for axis 1 and 2.
```

FOLMD Master Distance

Type	Following	Product	Rev
Syntax	<!><@><a>FOLMD<r>,<r>,<r>,<r>	AT6n00	3.0
Units	i = distance in steps	AT6n50	3.0
Range	i = 999,999,999 (scalable by SCLMAS)	610n	4.0
Default	0	615n	3.0
Response	FOLMD *FOLMD0,0,0,0	620n	3.0
	1FOLMD *FOLMD0	625n	3.0
See Also	FOLEN, FOLK, FOLRN, FOLRNF, FOLRD, MC, [PMAS], SCLMAS, TPMAS	6270	3.0

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

If scaling is enabled (SCALE1), the FOLMD value is specified in user units and is scaled by the SCLMAS parameter. Numeric variables (VAR) can be used with this command (e.g., FOLMD12,(VAR6),3,6).

By carefully specifying accurate master distances for each ramp of a slave's move profile, a precise position relationship between master and slave will be maintained during all phases of the profile. The "Master and Slave Distance Calculation" section in the Following chapter of the *6000 Series Programmer's Guide* discusses the relationship between ratio changes and the corresponding master and slave distances.

HINT: If a slave is in continuous mode (MC1) and the master is starting from rest, setting FOLMD to Ø will ensure precise tracking of the master's acceleration ramp. This is how the trackball application example is written in the Following chapter of the *6000 Series Programmer's Guide*.

Examples: (refer also to FOLEN example #2)

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

FOLRD Denominator of Slave-to-Master Ratio

Type	Following	Product	Rev
Syntax	<!><@><a>FOLRD<r>,<r>,<r>,<r>	AT6n00	3.0
Units	r = master distance in steps	AT6n50	3.0
Range	r = 1.00000 - 999,999,999 (scalable by SCLMAS)	610n	4.0
Default	1	615n	3.0
Response	FOLRD *FOLRD1,1,1,1	620n	3.0
	1FOLRD *FOLRD1	625n	3.0
See Also	COMEXC, FOLEN, FOLK, FOLMAS, FOLRN, FOLRNF, SCLMAS	6270	3.0

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

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

If scaling is enabled (SCALE1), the FOLRD value is scaled by the SCLMAS value.

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

Each time FOLRN or FOLRD are given, the 6000 controller divides the scaled numerator and denominator to calculate the ratio, but roundoff errors are eliminated by measuring both master and slave over a large distance. After scaling, the maximum magnitude of the ratio is 127 slave steps for every master step.

ON-THE-FLY CHANGES (as of revision 4.0): You can change Following ratio *on the fly* (while motion is in progress) in two ways. One way is to send an immediate command (!FOLRD) followed by an immediate go command (!GO). The other way is to enable the continuous command execution mode (COMEXC1) and execute a buffered command (FOLRD) followed by a buffered go command (GO).

Example: (refer also to the FOLEN examples)

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

FOLRN — Numerator of Slave-to-Master Ratio

		Product	Rev
Type	Following		
Syntax	<!><@><a>FOLRN<r>,<r>,<r>,<r>	AT6n00	3.0
Units	r = slave distance in steps	AT6n50	3.0
Range	r = 0.00000 - 999,999,999.99999 (scalable by SCLD)	610n	4.0
Default	1	615n	3.0
Response	FOLRN *FOLRN1,1,1,1 1FOLRN *FOLRN1	620n	3.0
		625n	3.0
See Also	FOLEN, FOLK, FOLMAS, FOLRNF, FOLRD, SCLD	6270	3.0

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

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

If scaling is enabled (SCALE1), the FOLRN value is scaled by the SCLD value.

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

Each time FOLRN or FOLRD are given, the 6000 controller divides the scaled numerator and denominator to calculate the ratio, but roundoff errors are eliminated by measuring both master and slave over a large distance. After scaling, the maximum magnitude of the ratio is 127 slave steps for every master step.

ON-THE-FLY CHANGES (as of revision 4.0): You can change Following ratio *on the fly* (while motion is in progress) in two ways. One way is to send an immediate command (!FOLRN) followed by an immediate go command (!GO). The other way is to enable the continuous command execution mode (COMEXC1) and execute a buffered command (FOLRN) followed by a buffered go command (GO).

Example: refer to the FOLRD and FOLEN examples

FOLRNF — Numerator of Final Slave-to-Master Ratio, Preset Moves

		Product	Rev
Type	Following; Compiled Motion		
Syntax	<!><@><a>FOLRNF<r>,<r>,<r>,<r>	AT6n00	4.1
Units	r = slave distance in steps	AT6n50	4.1
Range	0.00000	610n	4.0
Default	0	615n	4.1
Response	FOLRNF *FOLRNF1,1,1,1 1FOLRNF *1FOLRNF1	620n	4.1
		625n	4.1
See Also	FOLEN, FOLRD, FOLRN, FOLMD, SCLD,	6270	4.1

The Numerator of Final Slave-to-Master Ratio, Preset Moves (FOLRNF) command establishes the numerator of the final ratio between slave and master travel. (Ratios are always specified as positive, similar to velocity.) The FOLRNF command designates that the motor will move the load the distance designated in a preset GOBUF

segment, completing the move at a final ratio of zero. FOLRNF applies only to the first subsequent GOBUF, which marks an intermediate "end of move" within a following profile. FOLRNF is used only in conjunction with the GOBUF command. Normal preset GO moves always finish with zero FOLRNF.

If scaling is enabled (SCALE1), the FOLRNF value is scaled by the SCLD value.

For the Revision 4.0 release of the 6000 series, the only allowable value for FOLRNF is Ø, and it may only be used with compiled preset Following moves (a non-zero FOLRNF value will result in an immediate error message). FOLRNF is allowed for a segment only if the starting ratio is also zero, i.e., it must be the first segment, or the previous segment must have ended in zero ratio.

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

For more information on using the FOLRNF command, refer to the Custom Profiling chapter in the *6000 Series Programmer's Guide*.

FOLSND Following Step and Direction

Type	Following
Syntax	<!><@><a>FOLSND<b><b><b><b>
Units	n/a
Range	b = 0 (quadrature signal), 1 (step & direction) or X (don't care)
Default	0
Response	FOLSND: *FOLSND0000 1FOLSND: *1FOLSND0
See Also	CNTE, FOLEN

Product	Rev
AT6n00	4.5
OEM-AT6n00	n/a
AT6n50	4.5
610n	4.5
615n	4.5
620n	4.5
625n	4.5
6270	4.5

Use the FOLSND command to specify the functionality of the encoder input.

FOLSND0.....(default setting) accept a quadrature signal from an encoder

FOLSND1.....Accept step and direction signals. The count is registered on a positive edge of a transition for a signal measured on encoder channel A+ and A- connections. The direction of the count is specified by the signal on encoder channel B+ and B- connections. Therefore, you should connect your step and direction input device as follows: Connect Step+ to A+, Step- to A-, Direction+ to B-, and Direction- to B+. NOTE: The FOLSND1 feature may not be used while the Hardware Counter Input feature is enabled with CNTE1.

615n: This feature is applicable only to the external encoder input.

FPPEN Master Position Prediction Enable

Type	Following
Syntax	<!><@><a>FPPEN<b><b><b><b>
Units	n/a
Range	b = 0 (disable), 1 (enable) or X (don't change)
Default	1
Response	FPPEN *FPPEN0,0,0,0 1FPPEN *1FPPEN0
See Also	[FS], TFS

Product	Rev
AT6n00	3.0
AT6n50	3.0
610n	4.0
615n	3.0
620n	3.0
625n	3.0
6270	3.0

The FPPEN command enables or disables Master Position Prediction in the 6000 controller Following algorithm. Master Position Prediction is enabled by default, but can be disabled as desired with the FPPENØ command.

The 6000 controller measures master position once per *position sample period* and calculates a corresponding slave commanded position. This calculation, and achieving the subsequent slave commanded

position, requires 2 sample periods. (Steppers: position sample period is 2 ms. Servos: position sample period is determined by the current SSFR and INDAX values – see table in SSFR command description.)

Enabling Master Position Prediction (FPPEN1) eliminates any lag in slave position which would be dependent on master speed. It may be desirable to disable Master Position Prediction (FPPENØ) when maximum slave smoothness is important and minor phase delays can be accommodated. A detailed discussion of Master Position Prediction is given in the Following chapter of the *6000 Series Programmer's Guide.*

Example:
```
FPPEN11         ; Enable Master Position Prediction for axis 1 and 2.
```

FR — Feedrate Override Enable

Type	Feedrate Override
Syntax	<!>FR<i>
Units	n/a
Range	i = 0 (disable), 1 (analog), or 2 (software)
Default	0
Response	FR: *FR0
See Also	FRA, FRH, FRPER, JOYEDB

Product	Rev
AT6n00	1.0
AT6n50	n/a
610n	n/a
615n	n/a
620n	1.0
625n	n/a
6270	n/a

The Feedrate Override Enable (FR) command enables feedrate override on all axes. The feedrate override percentage can be determined either through hardware (except OEM-AT6n00) or through software (FRPER command).

To use the analog inputs to control the feedrate override percentage, use the FR1 command. With the FR1 command, the channel number specified is used in the FRH command or the FRL command, depending on the level of the channel select input, to determine which analog channel will scale the motion velocity. The analog input can go from ØVDC to 2.5VDC. The motion velocity will be scaled by the percentage of analog input voltage, ØVDC equaling Ø%, 2.5VDC equaling 100%. The JOYEDB command will set the end deadband for feedrate override.

NOTES

Timer Functions Scaled: All timer functions will be scaled when feedrate override is enabled. For example, a T5 command at a 50% feedrate (FRPER5Ø) will dwell for 10 seconds.

Feedrate Override While Contouring: If you change the FR command setting, you will have to recompile (PCOMP) any previously compiled contouring paths.

WARNING

When using feedrate override on a four-axis 6000 controller, axis 4 is used to perform the feedrate override and can no longer be used for motion. If the shutdown output is not used, you must disconnect axis 4; otherwise, motion will occur on that axis.

To use the software feedrate override percentage (FRPER), specify FR2.

Example:
```
FRL3              ; When the channel select input is low, use analog input #3
FRH4              ; When the channel select input is high, use analog input #4
SCALE1            ; Enable scaling
SCLA25000,25000   ; Set accel scaling factor to 25000 steps/unit/unit on axes
                  ; 1 and 2
FRA1000           ; Set the feedrate override acceleration to
                  ; 1000 percent/sec/sec
FR1               ; Enable analog feedrate override
V20,20            ; Set the velocity to 20 units/sec on axes 1 and 2
MC1100            ; Mode continuous on axis 1 and axis 2
GO1100            ; Initiate motion on axes 1 and 2
; If Channel Select input is low, and voltage on analog input 3 is
; 2.0VDC, then velocity for axes 1 & 2 will be:
; 20 units/sec x 2.0VDC/2.5VDC = 16 units/sec.
```

FRA — Feedrate Override Acceleration

Type	Feedrate Override
Syntax	<!>FRA<r>
Units	r = percent/sec/sec
Range	1 - 50000
Default	10
Response	FRA: *FRA10
See Also	FR, FRH, FRL, FRPER

Product	Rev
AT6n00	1.4
AT6n50	n/a
610n	n/a
615n	n/a
620n	1.5
625n	n/a
6270	n/a

The Feedrate Override Acceleration (FRA) command specifies the acceleration and deceleration to use when the velocity is changing due to a change in voltage on one of the analog inputs (FR1), or when the software feedrate override percentage (FRPER) is changing (FR2).

Since the maximum value for the feedrate is 100% per second and the update is 2 ms, the maximum value for feedrate acceleration is 50000 %/sec/sec.

Example: Refer to the feedrate override enable (FR) command example.

FRH — Feedrate Override High Channel

Type	Feedrate Override
Syntax	<!>FRH<i>
Units	i = analog input channel number (varies with product)
Range	0 (no selection), 1 (input 1), 2 (input 2), 3 (input 3), or 4 (input 4)
Default	0
Response	FRH: *FRH0
See Also	FR, FRA, FRL

Product	Rev
AT6n00	1.0
OEM-AT6n00	n/a
AT6n50	n/a
610n	n/a
615n	n/a
620n	1.0
625n	n/a
6270	n/a

The Feedrate Override High Channel (FRH) command specifies which analog input channel will be used when the axis select input (pin 15 of joystick connector) is high. The 6000 Series product will not use any channel if FRH is set to zero, instead it will operate at the current velocity (V value).

When feedrate override is enabled for analog control (FR1), the channel number specified in the FRH command or the FRL command (depending on the level of the axis select input) is used to determine which analog channel will scale the motion velocity. The analog input can go from ØVDC to 2.5VDC. The motion velocity will be scaled by the percentage of analog input voltage, ØVDC equaling Ø%, 2.5VDC equaling the value specified in the V command.

Example: Refer to the feedrate override enable (FR) command example.

FRL — Feedrate Override Low Channel

Type	Feedrate Override
Syntax	<!>FRL<i>
Units	i = analog input channel number (varies with product)
Range	0 (no selection), 1 (input 1), 2 (input 2), 3 (input 3), or 4 (input 4)
Default	0
Response	FRL: *FRL0
See Also	FR, FRA, FRH

Product	Rev
AT6n00	1.0
OEM-AT6n00	n/n
AT6n50	n/n
610n	n/a
615n	n/n
620n	1.0
625n	n/n
6270	n/n

The Feedrate Override Low Channel (FRL) command specifies which analog input channel will be used when the axis select input (pin 15 of joystick connector) is low. The 6000 Series product will not use any channel if FRL is set to zero, instead it will operate at the current velocity (V value).

When feedrate override is enabled for analog control (FR1), the channel number specified in the FRH command or the FRL command (depending on the level of the axis select input) is used to determine which analog channel will scale the motion velocity. The analog input can go from ØVDC to 2.5VDC. The motion velocity will be scaled by the percentage of analog input voltage, ØVDC equaling Ø%, 2.5VDC equaling the value specified in the V command.

Example: Refer to the feedrate override enable (FR) command example.

FRPER Feedrate Override Percentage

Type	Feedrate Override
Syntax	<!>FRPER<r>
Units	r = percent
Range	0 - 100
Default	100
Response	FRPER: *FRPER100
See Also	FR, FRA

Product	Rev
AT6n00	1.0
AT6n50	n/a
610n	n/a
615n	n/a
620n	1.0
625n	n/a
6270	n/a

The Feedrate Override Percentage (FRPER) command specifies the percentage by which motion velocity will be scaled when feedrate override is enabled (FR2). The percentage range available to scale the feedrate override by is Ø% to 1ØØ%, or in other words Ø% of the current velocity to 1 times the current velocity (V).

Example:

```
FRA1000          ; Set the feedrate override acceleration to 1000 percent/sec/sec
FRPER90          ; Set the feedrate override percentage at 90%
FR2              ; Enable feedrate override
V20,20           ; Set the velocity to 20 units/sec on axes 1 and 2
MC1100           ; Mode continuous on axis 1 and axis 2
GO1100           ; Initiate motion on axes 1 and 2.
                 ; The velocity for axes 1 and 2 will be:
                 ; 20 units/sec x .90 = 18 units/sec
```

[FS] Following Status

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

Product	Rev
AT6n00	3.0
AT6n50	3.0
610n	4.0
615n	3.0
620n	3.0
625n	3.0
6270	3.0

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

Syntax: VARBn=aFS where n is the binary variable number and a is the axis identifier, or FS can be used in an expression such as IF(1FS=b11Ø1), or IF(1FS=h7F). The FS command must be used with an axis specifier, or it will default to axis 1.

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

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

Bit Assignment (left to right)	Function (YES = 1; NO = Ø)	
1	Slave in Ratio Move	A Following move is in progress.
2	Ratio is Negative	The current ratio is negative (i.e., the slave counts are counting in the opposite direction from the master counts).
3	Slave Ratio Changing	The slave is ramping from one ratio to another (including a ramp to or from zero ratio).
4	Slave At Ratio	The slave is at constant non-zero ratio.
* 5	FOLMAS Active	A master is specified with the FOLMAS command.
* 6	FOLEN Active	Following has been enabled with the FOLEN command.
* 7	Master is Moving	The specified master is currently in motion.
8	Master Dir Neg	The current master direction is negative. (bit must be cleared to allow Following move in preset mode–MCØ).
9	OK to Shift	Conditions are valid to issue shift commands (FSHFD or FSHFC).
10	Shifting now	A shift move is in progress.
11	Shift is Continuous	An FSHFC-based shift move is in progress.
12	Shift Dir is Neg	The direction of the shift move in progress is negative.
13	Master Cyc Trig Pend	A master cycle restart is pending the occurrence of the specified trigger.
14	Mas Cyc Len Given	A non-zero master cycle length has been specified with the FMCLEN command.
15	Master Cyc Pos Neg	The current master cycle position (PMAS) is negative. This could be by caused by a negative initial master cycle position (FMCP), or if the master is moving in the negative direction.
16	Master Cyc Num > 0	The master position (PMAS) has exceeded the master cycle length (FMCLEN) at least once, causing the master cycle number (NMCY) to increment.
17	Mas Pos Prediction On	Master position prediction has been enabled (FPPEN).
18	Mas Filtering On	A non-zero value for master position filtering (FFILT) is in effect.

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

Example:

```
VARB1=1FS       ; Following status for axis 1 assigned to binary variable 1
VARB2=1FS.12    ; Axis 1 Following status bit 12 assigned to binary variable 2
VARB2           ; Response if bit 12 is set to 1 should be:
                ; *VARB2=XXXX_XXXX_XXX1_XXXX_XXXX_XXXX_XXXX_XXXX
IF(4FS=b111011X11)  ; If the Following status for axis 4 contains 1's for
                ; inputs 1, 2, 3, 5, 6, 8, and 9, and a 0 for bit location 4,
                ; do the IF statement
TREV            ; Transfer revision level
NIF             ; End if statement
IF(2FS=h7F00)   ; If the Following status for axis 2 contains 1's for inputs 1,
                ; 2, 3, 5, 6, 7, and 8, and 0's for every other bit location,
                ; do the IF statement
TREV            ; Transfer revision level
NIF             ; End if statement
```

FSHFC Continuous Shift

		Product	Rev
Type	Following		
Syntax	<!><@><a>FSHFC<i>,<i>,<i>,<i>	AT6n00	3.0
Units	i = shift feature to implement	AT6n50	3.0
Range	i = 0 (stop), 1 (positive-direction), 2 (negative-direction), or 3 (kill)	610n	4.0
Default	n/a	615n	3.0
Response	n/a	620n	3.0
See Also	FOLEN, FOLK, FOLRN, FOLRNF, FOLRD, [FS], FSHFD, MC, [PSHF], TFS, TPSHF	625n	3.0
		6270	3.0

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

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

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

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

An FSHFC move may be needed to adjust the relative slave position on the fly during the continuous Following move. For example, suppose an operator is visually inspecting the slave's motion with respect to the master. If he notices that the master and slave are out of synchronization, it may be desirable to have an interrupt programmed (e.g., activated with a push-button switch) that will allow the operator to advance or retard the slave at a super-imposed correction speed until the operator chooses to have the slave start tracking the master again. The example below shows this.

FSHFC Example:

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

(Program on following page)

Example:

```
DEF SHIFT       ; Begin definition of program called SHIFT
V.1             ; Add or subtract 0.1 user scaled units from the slave velocity
                ; when shifting
INFEN1          ; Enable input functions
COMEXS1         ; Continue command execution after stop
COMEXC1         ; Continue command execution during motion
FOLMAS31        ; Axis 3 encoder input is the master for axis 1
FOLRN1          ; Set slave-to-master Following ratio numerator to 1
FOLRD1          ; Set slave-to-master Following ratio denominator to 1
                ; (ratio set to 1:1)
FOLEN1          ; Enable Following mode on axis #1
D+              ; Set to positive-direction
MC1             ; Select continuous positioning mode
GO1             ; Start following master continuously
VARB1=b1Ø       ; Define input pattern #1 and assign to VARB1
VARB2=bØ1       ; Define input pattern #2 and assign to VARB2
$TESTIN         ; Define label called TESTIN
IF(IN=VARB1)    ; IF statement (if input #1 is activated, do the jump)
  JUMP SHIFTP   ; Jump to shift slave in the positive-direction when pattern 1
                ; active
  NIF           ; End of IF statement
IF(IN=VARB2)    ; IF statement (if input #2 is activated, do the jump)
  JUMP SHIFTN   ; Jump to shift slave in the negative-direction when pattern 2
                ; active
  NIF           ; End of IF statement
JUMP TESTIN     ; Return to main program loop
$SHIFTP         ; Define label called SHIFTP (subroutine to shift in the
                ; positive- direction)
FSHFC1          ; Start continuous slave shift move in positive-direction
WAIT(IN.1=bØ)   ; Continue shift until input bit #1 is deactivated
FSHFCØ          ; Stop shift move
JUMP TESTIN     ; Return to main program loop
$SHIFTN         ; Define label called SHIFTN (subroutine to shift in the
                ; negative-direction)
FSHFC2          ; Start continuous slave shift move in the negative-direction
WAIT(IN.2=bØ)   ; Continue shift until bit #2 is deactivated
FSHFCØ          ; Stop shift move
JUMP TESTIN     ; Return to main program loop
END             ; End definition of program called SHIFT
```

FSHFD — Preset Shift

		Product	Rev
Type	Following		
Syntax	<!><@><a>FSHFD<r>,<r>,<r>,<r>	AT6n00	3.0
Units	r = shift distance	AT6n50	3.0
Range	r = 0.00000 - 999,999,999 (scalable with SCLD)	610n	4.0
Default	n/a	615n	3.0
Response	n/a	620n	3.0
See Also	FOLEN, FOLK, FOLRN, FOLRNF, FOLRD, [FS], FSHFC, MC, ONCOND, [PSHF], SCLD, TFS, TPSHF	625n	3.0
		6270	3.0

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

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

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

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

It should be noted that FSHFD is similar in execution to GO. The entire preset distance shift, or ramp-to-shift velocity, must finish before the 6000 controller proceeds to the next command.

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

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

FSHFD **Example:**

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

Example:

```
DEF PSHIFT      ; Begin definition of program called PSHIFT
INFEN1          ; Enable input functions
COMEXS1         ; Continue command execution after stop
COMEXC1         ; Continue command execution during motion
FOLMAS31        ; Axis 3 encoder input is the master for axis 1
FOLRN1          ; Set slave-to-master Following ratio numerator to 1
FOLRD1          ; Set slave-to-master Following ratio denominator to 1
                ; (ratio set to 1:1)
FOLEN1          ; Enable Following mode on axis #1
D+              ; Set direction to positive
MC1             ; Select continuous positioning mode
GO1             ; Start following master continuously
VARB1=b1Ø       ; Define input pattern #1 and assign to VARB
VARB2=bØ1       ; Define input pattern #2 and assign to VARB
$TESTIN         ; Define label called TESTIN
IF(IN=VARB1)    ; IF statement (if input #1 is activated, do the jump)
  JUMP SHIFTP   ; Jump to shift slave in the positive-direction when pattern 1
                ; active
  NIF           ; End of IF statement
IF(IN=VARB2)    ; IF statement (if input #2 is activated, do the jump)
  JUMP SHIFTN   ; Jump to shift slave in the negative-direction when pattern 2
                ; active
  NIF           ; End of IF statement
JUMP TESTIN     ; Return to main program loop
$SHIFTP         ; Define label called SHIFTP (subroutine to shift in the
                ; positive direction)
FSHFD2ØØ        ; Start preset slave shift move of 200 steps in the positive
                ; direction
WAIT(FS.1Ø=bØ)  ; Wait for shift to finish
JUMP TESTIN     ; Return to main program loop
$SHIFTN         ; Define label called SHIFTN (subroutine to shift in the
                ; negative direction)
FSHFD-2ØØ       ; Start preset slave shift move of 200 steps in the negative
                ; direction
WAIT(FS.1Ø=bØ)  ; Wait for shift to finish
JUMP TESTIN     ; Return to main program loop
END             ; End definition of program called PSHIFT
```

GO — Initiate Motion

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

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

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

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

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

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

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

- Acceleration and S-curve Acceleration (A and AA)
- Deceleration and S-curve Deceleration (AD and ADA)
- Velocity (V)
- Distance (D)
- Preset or Continuous Positioning Mode Selection (MC)
- Incremental or Absolute Positioning Mode Selection (MA)
- Following Ratio Numerator and Denominator (FOLRN and FOLRD, respectively)

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

For more information, refer to the Custom Profiling section in the *6000 Series Programmer's Guide*.

Example:

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

GOBUF Store a Motion Segment in Compiled Memory

		Product	Rev
Type	Compiled Motion	AT6n00	4.1
Syntax	<@>GOBUF<b><b><b><b>	AT6n50	4.1
Units	n/a	610n	4.0
Range	b = 0 (don't go), 1 (go), or X (don't change)	615n	4.1
Default	1	620n	4.1
Response	n/a	625n	4.1
See Also	[AS], DEF, END, [ER], FOLRNF, MA, MC, MEMORY, PCOMP, POUTn, PRUN, PUCOMP, PLOOP, PLN, [SS], TAS, TER, TSS, VF	6270	4.1

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

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

Each GOBUF motion segment may have its own distance to travel, velocity, acceleration and deceleration. The end of a preset segment (MCØ) is determined by the distance or position specified; a compiled MCØ GOBUF motion segment is finished when the "D" goal is reached. The end of a continuous segment (MC1) is determined by the ratio or velocity specified; a compiled MC1 GOBUF motion segment is finished when the velocity or ratio goal is reached. If either a preset segment or continuous segment is followed by a compiled GOWHEN command, motion will continue at the last velocity until the GOWHEN condition becomes true, and the next segment begins.

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

Starting velocity of a GOBUF segment

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

Ending velocity of a GOBUF segment

Preset Positioning Mode (MCØ)

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

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

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

Continuous Positioning Mode (MC1)

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

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

NOTE: Each GOBUF motion segment can consume from 2-8 memory segments of compiled memory. If

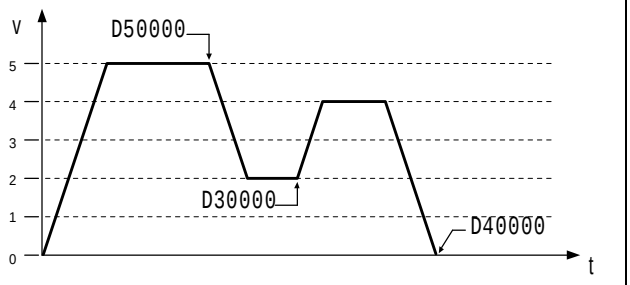
there is no more space left in compiled memory, a compilation error will result.

Example:

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

PCOMP simple ; Compile simple
PRUN simple  ; Run simple
```

The resulting profile from this program:



GOL — Initiate Linear Interpolated Motion

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

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	n/a
615n	n/a
620n	1.0
625n	1.0
6270	1.0

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

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

When moves are made using the GOL command, the endpoint of the linear interpolated move is determined by the D command. The accelerations, decelerations, and velocities for the individual axes are calculated internally by the 6000 Series product, so that the load is moved *in a straight line* at the path acceleration (PA and PAD) and velocity entered (PV). In other words, the path acceleration (PA), path average acceleration (PAA), the path deceleration (PAD), path average deceleration (PADA), and the path velocity (PV) all correspond to the rate of travel required to go to the point in space specified by the D command. All axes are to arrive at the same time; therefore, if each axis' distance is different, each axis must travel at a different rate to have each axis arrive at the same time. The 6000 Series product takes care of the calculations for each axis, you just enter the overall rate of travel.

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

Example:

```
SCALE1          ; Enable scaling
PSCLA25000      ; Set path acceleration scale factor to 25000 steps/unit/unit
PSCLV25000      ; Set path velocity scale factor to 25000 steps/unit
@SCLD10000      ; Set distance scale factor to 10000 steps/unit on all axes
PA25            ; Set the path acceleration to 25 units/sec/sec
PAD20           ; Set the path deceleration to 20 units/sec/sec
PV2             ; Set the path velocity to 2 units/sec
D10,5,2,11      ; Set the distance to 10, 5, 2, and 11 units on axes 1-4
GOL1111         ; Initiate linear interpolated motion on all axes. A GOL command
                ; could have been issued instead of a GOL1111 command.
```

GOSUB — Call a Subroutine

		Product	Rev
Type	Program or Subroutine Definition or Program Flow Control	AT6n00	1.0
Syntax	<!>GOSUB<t>	AT6n50	1.0
Units	t = text (name of program/subroutine)	610n	4.0
Range	Text name of 6 characters or less	615n	1.0
Default	n/a	620n	1.0
Response	n/a	625n	1.0
See Also	$, BREAK, DEF, DEL, END, ERASE, GOTO, JUMP, RUN	6270	1.0

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

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

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

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

Example:

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

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

GOTO — Goto a Program or Label

		Product	Rev
Type	Program or Subroutine Definition or Program Flow Control	AT6n00	1.0
Syntax	<!>GOTO<t>	AT6n50	1.0
Units	t = text (name of program/label)	610n	4.0
Range	Text name of 6 characters or less	615n	1.0
Default	n/a	620n	1.0
Response	n/a	625n	1.0
See Also	$, DEF, DEL, END, GOSUB, IF, JUMP, L, LN, NIF, NWHILE, REPEAT, RUN, UNTIL, WHILE	6270	1.0

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

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

CAUTION

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

Example:

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

GOWHEN Conditional Go

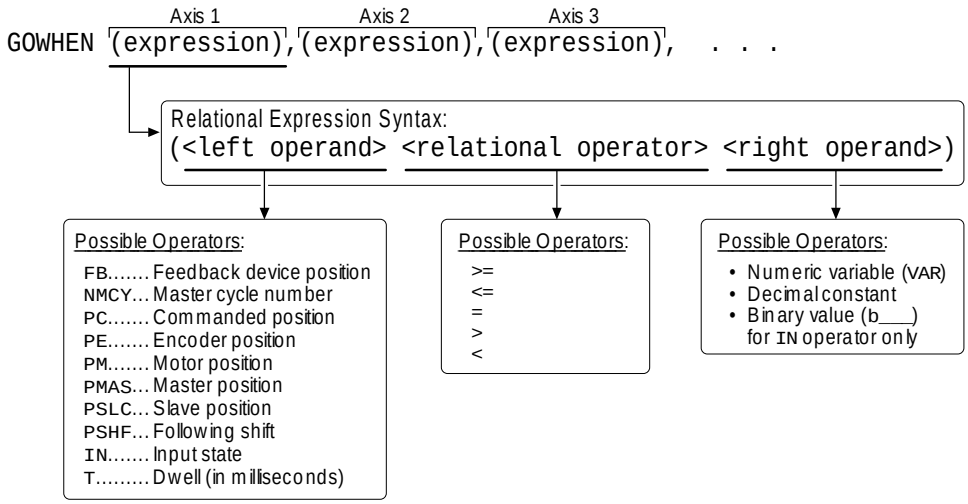
Type	Motion; Following
Syntax	<!><@><a>GOWHEN(expression,expression,expression,expression)
Units	n/a
Range	Up to 80 characters (including parentheses)
Default	n/a
Response	n/a
See Also	[AS], COMEXC, [ER], ERROR, ERRORP, [FB], FSHFC, FSHFD, GO, GOL, [IN], [NMCY], [PC], [PE], [PM], [PMAS], [PSHF], [PSLV], T, TAS, TER, TRGFN, WAIT

Product	Rev
AT6n00	3.0
AT6n50	3.0
610n	4.0
615n	3.0
620n	3.0
625n	3.0
6270	3.0

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

Start-motion type commands that **cannot** be synchronized using the GOWHEN command are: HOM, JOG, JOY, and PRUN. A preset GO command that is already in motion can start a new profile using the GOWHEN and GO sequence of commands. Continuous moves (MC1) already in progress can change to a new velocity based upon the GOWHEN and GO sequence. Both preset and continuous moves can be started from rest with the GOWHEN and GO sequence.

GOWHEN Syntax:



EXAMPLES

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

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

GOWHEN status:

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

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

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

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

GOWHEN ... On a Trigger Input:

If you wish motion to be triggered with a trigger input, use the TRGFNc1xxxxxxx command. The TRGFNc1xxxxxxx command executes in the same manner as the GOWHEN command, except that motion is executed when the specified trigger input (c) is activated. For more information, refer to the TRGFN command description.

GOWHEN vs. WAIT:

A WAIT will cause the 6000 controller program to halt program flow (except for execution of immediate commands) until the condition specified is satisfied. Common uses for this function include delaying subsequent I/O activation until the master has achieved a required position or an object has been sensed.

By contrast, a GOWHEN will suspend the motion profile for a specific axis until the specified condition is met. It does **not** affect program flow. If you wish motion to be triggered with a trigger input, use the TRGFNc1xxxxxxx command. The TRGFNc1xxxxxxx command executes in the same manner as the GOWHEN command, except that motion is executed when the specified trigger input (c) is activated (see TRGFN command description for details). In addition, GOWHEN expressions are limited to the operands listed above; WAIT can use additional operands such as FS (Following status) and VMAS (velocity of master).

Factors Affecting GOWHEN Execution:

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

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

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

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

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

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

Sample 6000 Code:

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

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

[h] Hexadecimal Identifier

		Product	Rev
Type	Operator (Other)		
Syntax	See Below	AT6n00	1.0
Units	n/a	AT6n50	1.0
Range	n/a	610n	4.0
Default	n/a	615n	1.0
Response	n/a	620n	1.0
See Also	[AS], [b], [ER], [IN], [INO], [LIM], [MOV], [OUT], [SS], [US], VARB	625n	1.0
		6270	1.0

This identifier allows you to specify hexadecimal values. A capital h (H) is valid as well. All other bits not specified are set to zero.

Example:

```
VARB1=h32FD     ; Set binary variable #1 to hex 32FD
```

HALT Terminate Program Execution

		Product	Rev
Type	Program Flow Control		
Syntax	<!>HALT	AT6n00	1.0
Units	n/a	AT6n50	1.0
Range	n/a	610n	4.0
Default	n/a	615n	1.0
Response	n/a	620n	1.0
See Also	BP, BREAK, C, ELSE, IF, K, NIF, NWHILE, PS, REPEAT, S, T, UNTIL, WAIT, WHILE	625n	1.0
		6270	1.0

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

Example:

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

HELP — Technical Support

Type	Program Debug Tool
Syntax	<!>HELP
Units	n/a
Range	n/a
Default	n/a
Response	See description below
See Also	None

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

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

HOM — Go Home

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

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

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

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

NOTE

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

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

- Home acceleration (HOMA and HOMAA)
- Home deceleration (HOMAD and HOMADA)
- Home velocity (HOMV)
- Final home velocity (HOMVF)
- Home reference edge (HOMEDG)
- Backup to home (HOMBAC)
- Final home direction (HOMDF)
- Active state of home input (HOMLVL)
- Home to encoder Z-channel (HOMZ) – n/a to OEM-AT6n00

Steppers: All homing parameters are valid in either motor step mode (ENCØ) or encoder step mode (ENC1).

For more information on homing refer to *Homing* section of the *6000 Series Programmer's Guide*.

Example:

```
@MA0            ; Incremental index mode for all axes
@MC0            ; Preset index mode for all axes
SCALE1          ; Enable scaling
SCLA25000,25000,1,1 ; Set accel. scaling: axes 1 & 2 = 25000 steps/unit/unit;
                ; axes 3 & 4 = 1 step/unit/unit
SCLV25000,25000,1,1 ; Set vel. scaling: axes 1 & 2 = 25000 steps/unit;
                ; axes 3 & 4 = 1 step/unit
@SCLD1          ; Set distance scaling factor for all axes to 1 step/unit
HOMA10,12,1,2   ; Set home acceleration to 10, 12, 1, & 2 units/sec/sec for
                ; axes 1, 2, 3 & 4
@HOMAD20        ; Set home deceleration to 20 units/sec/sec for all axes
HOMBAC1100      ; Enable backup to home switch on axes 1 and 2 only
HOMEDG0011      ; Axes 1 & 2 stop on the positive-direction edge of the home
                ; switch, axes 3 and 4 are to stop on negative-direction side
@HOMDF0         ; Set final home direction to positive on all axes.
@HOMZ0          ; Disable homing to encoder Z-channel on all axes
@HOMLVL0        ; Set home active level to low on all axes
HOMV1,1,1,2     ; Set home velocity to 1, 1, 1, and 2 units/sec for
                ; axes 1, 2, 3 & 4
@HOMVF.1        ; Sets home final velocity to 0.1 units/sec for all axes
HOM01XX         ; Execute go home in positive-direction on axis 1,
                ; negative-direction on axis 2. Do not home on axes 3 and 4.
```

HOMA — Home Acceleration

Type	Homing
Syntax	<!><@><a>HOMA<r>,<r>,<r>,<r>
Units	r = units/sec/sec
Range	0.00025 - 24,999,999 (depending on the scaling factor)
Default	10.0000
Response	HOMA: *HOMA10.0000,10.0000,10.0000,10.0000 1HOMA: *1HOMA10.0000
See Also	HOM, HOMAD, HOMBAC, HOMDF, HOMEDG, HOMLVL, HOMV, HOMVF, HOMZ, SCALE, SCLA

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

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

Steppers: The entered value is always in reference to motor steps, not encoder steps, regardless of the ENC command setting. If scaling is not enabled (SCALEØ), the acceleration value is entered in motor revs/sec/sec; this value is internally multiplied by the drive resolution (DRES) value to obtain an acceleration value in motor steps/sec/sec for the motion trajectory calculations. If scaling is enabled (SCALE1), the entered acceleration value is internally multiplied by the acceleration scaling factor (SCLA) to convert user units/sec/sec to motor steps/sec/sec.

Servos: If scaling is not enabled (SCALEØ), the acceleration value is entered in encoder or resolver revs/sec/sec, LDT inches/sec/sec, or ANI volts/sec/sec; encoder or resolver and LDT values are internally multiplied by the encoder resolution (ERES) value or the LDT resolution (LDTRES) value to obtain an acceleration value in steps/sec/sec for the motion trajectory calculations. If scaling is enabled (SCALE1), the entered acceleration value is internally multiplied by the acceleration scaling factor (SCLA) to convert user units/sec/sec to encoder, resolver, LDT, or ANI steps/sec/sec.

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

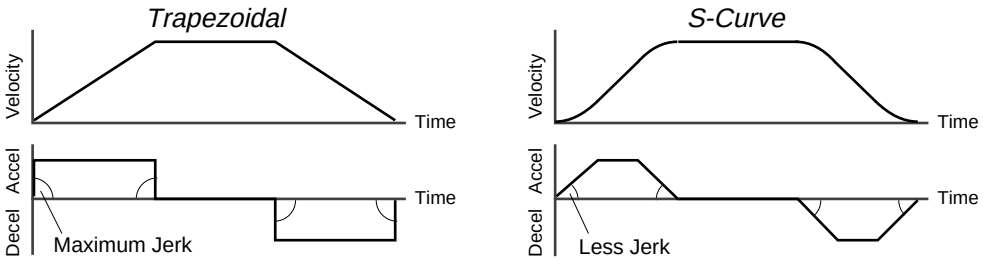
If the home deceleration (HOMAD) command has not been entered, the home acceleration (HOMA) command will set the home deceleration rate. Once the home deceleration (HOMAD) command has been entered, the home acceleration (HOMA) command no longer affects home deceleration.

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

HOMAA Homing Average Acceleration

		Product	Rev
Type	Motion (S-Curve)		
Syntax	<!><@><a>HOMAA<r>,<r>,<r>,<r>	AT6n00	n/a
Units	r = units/sec/sec	AT6n50	1.0
Range	0.00025 - 24999999 (depending on the scaling factor)	610n	n/a
Default	10.00 (trapezoidal profiling is default, where HOMAA tracks HOMA)	615n	1.0
Response	HOMAA: *HOMAA10.0000,10.0000,10.0000,10.0000 1HOMAA: *1HOMAA10.0000	620n	n/a
		625n	1.0
See Also	A, AD, ADA, HOM, HOMA, HOMAD, HOMADA, HOMBAC, SCALE, SCLA	6270	1.0

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



The values for the maximum homing accel (HOMA) and average homing accel (HOMAA) commands determine the characteristics of the S-curve. To smooth the acceleration ramp, you must enter a HOMAA command value that satisfies this equation: 1/2 HOMA ≤ HOMAA < HOMA. The following conditions are possible:

Acceleration Setting	Profiling Condition
HOMAA > 1/2 HOMA, but HOMAA < HOMA	S-curve profile with a variable period of constant acceleration
HOMAA = 1/2 HOMA	*Pure S-curve* (no period of constant acceleration—smoothest motion)
HOMAA = HOMA	Trapezoidal profile (but can be changed to an S-curve by specifying a new HOMAA value < HOMA)
HOMAA < 1/2 HOMA; or HOMAA > HOMA	When you issue the HOM command, the move will not be executed and an error message, *INVALID CONDITIONS FOR S_CURVE ACCELERATION–FIELD *n*, will be displayed.
HOMAA = zero	S-curve profiling is disabled. Trapezoidal profiling is enabled. HOMAA tracks HOMA, & HOMADA tracks HOMAD. (*Track* means the command's value will match the other command's value and will continue to match whatever the other command's value is set to.)
No HOMAA value ever entered	Profile will default to trapezoidal. HOMAA tracks HOMA.

While programming S-curves, if you never change the maximum or average homing deceleration (HOMAD or HOMADA) commands, HOMADA will track HOMAA. However, once you change HOMAD, HOMADA will no longer track changes in HOMAA.

NOTE

Once you enter a HOMAA value that is ≠ zero or ≠ HOMA, S-curve profiling is enabled **only for homing moves** (e.g., not for contouring, which requires the PADA and/or PAA commands). All subsequent homing moves for that axis must comply with this equation: 1/2 HOMA ≤ HOMAA < HOMA.

Increasing the AA value above the pure S-curve level (HOMAA > 1/2 HOMA), the time required to reach the target velocity and the target distance is decreased. However, increasing HOMAA also increases jerk.

The calculation for determining S-curve average accel and decel move times is as follows:

(A_{avg} = average accel or decel value):

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

Scaling (SCLA) affects HOMAA the same as it does for HOMA.

*** For a more in-depth discussion on S-curve profiling, refer to the Custom Profiling chapter in the *6000 Series Programmer's Guide*.

Example:

```
SCALE0          ; Disable scaling
@MA0            ; Select incremental positioning mode
HOMA10,10       ; Set homing max. accel to 10 rev/sec/sec (axes 1 and 2)
HOMAA5,10       ; Set homing avg. accel to 5 rev/sec/sec on axis 1,
                ; and 10 rev/sec/sec on axis 2
HOMAD10,10      ; Set homing max. decel to 10 rev/sec/sec (axes 1 and 2)
HOMADA5,10      ; Set homing avg. decel to 5 rev/sec/sec on axis 1,
                ; and 10 rev/sec/sec on axis 2
HOM11XX         ; Execute negative-direction homing moves on axes 1 and 2
; Axis 1 executes a pure S-curve; axis 2 executes a trapezoidal profile.
```

HOMAD Home Deceleration

		Product	Rev
Type	Homing	AT6n00	1.0
Syntax	<!><@><a>HOMAD<r>,<r>,<r>,<r>	AT6n50	1.0
Units	r = units/sec/sec	610n	4.0
Range	0.00025 - 24,999,999 (depending on the scaling factor)	615n	1.0
Default	10.0000 (HOMAD tracks HOMA)	620n	1.0
Response	HOMAD: *HOMAD10.0000,10.0000,10.0000,10.0000 1HOMAD: *1HOMAD10.0000	625n	1.0
See Also	HOM, HOMA, HOMAA, HOMADA, HOMBAC, HOMEDG, HOMDF, HOMLVL, HOMV, HOMVF, HOMZ, SCALE, SCLA	6270	1.0

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

Steppers: The entered value is always in reference to motor steps, not encoder steps, regardless of the ENC command setting. If scaling is not enabled (SCALEØ), the deceleration value is entered in motor revs/sec/sec; this value is internally multiplied by the drive resolution (DRES) value to obtain an deceleration value in motor steps/sec/sec for the motion trajectory calculations. If scaling is enabled (SCALE1), the entered deceleration value is internally multiplied by the deceleration scaling factor (SCLA) to convert user units/sec/sec to motor steps/sec/sec.

Servos: If scaling is not enabled (SCALEØ), the deceleration value is entered in encoder or resolver revs/sec/sec, LDT inches/sec/sec, or ANI volts/sec/sec; encoder or resolver and LDT values are internally multiplied by the encoder resolution (ERES) value or the LDT resolution (LDTRES) value to obtain an acceleration value in steps/sec/sec for the motion trajectory calculations. If scaling is enabled (SCALE1), the entered deceleration value is internally multiplied by the deceleration scaling factor (SCLA) to convert user units/sec/sec to encoder, resolver, LDT, or ANI steps/sec/sec.

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

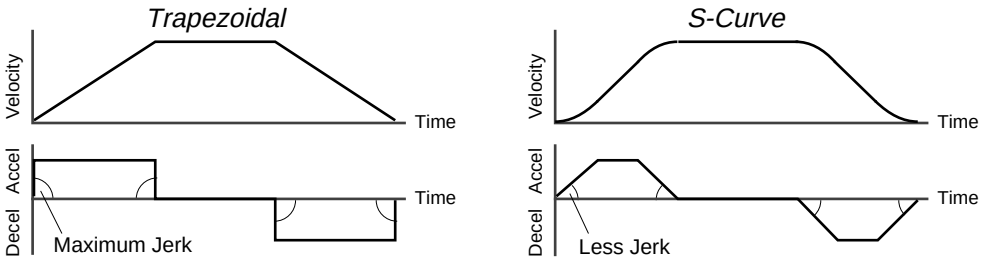
If the home deceleration (HOMAD) command has not been entered, the home acceleration (HOMA) command will set the deceleration rate. Once the home deceleration (HOMAD) command has been entered, the home acceleration (HOMA) command no longer affects home deceleration. If the HOMAD command is set to zero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

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

HOMADA Homing Average Deceleration

		Product	Rev
Type	Motion (S-Curve)	AT6n00	n/a
Syntax	<!><@><a>HOMADA<r>,<r>,<r>,<r>	AT6n50	1.0
Units	r = units/sec/sec	610n	n/a
Range	0.00025 - 24999999 (depending on the scaling factor)	615n	1.0
Default	10.00 (HOMADA tracks HOMAA)	620n	n/a
Response	HOMADA: *HOMADA10.0000,10.0000,10.0000,10.0000 1HOMADA: *1HOMADA10.0000	625n	1.0
See Also	A, AD, HOM, HOMA, HOMAA, HOMAD, SCALE, SCLA	6270	1.0

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



The values for the maximum homing decel (HOMAD) and average homing decel (HOMADA) commands determine the characteristics of the S-curve. To smooth the deceleration ramp, you must enter a HOMADA command value that satisfies this equation: 1/2 HOMAD ≤ HOMADA < HOMAD. The following conditions are possible:

Deceleration Setting	Profiling Condition
HOMADA > 1/2 HOMAD, but HOMADA < HOMAD	S-curve profile with a variable period of constant deceleration
HOMADA = 1/2 HOMAD	*Pure S-curve* (no period of constant deceleration—smoothest motion)
HOMADA = HOMAD	Trapezoidal profile (but can be changed to S-curve by specifying a new HOMADA value < HOMAD)
HOMADA < 1/2 HOMAD; or HOMADA > HOMAD	When you issue the HOM command, the move will not be executed and an error message, *INVALID CONDITIONS FOR S_CURVE ACCELERATION–FIELD *n*, will be displayed.
HOMADA = zero	Upon entering the HOMADAØ command, an error message, *INVALID DATA–FIELD *n*, will be displayed.
S-curve profiling with HOMAA, and no HOMADA or HOMAD ever entered	HOMADA will always match the HOMAA command value (identical S-curve accel and decel profiles). When you change HOMAD, HOMADA will no longer match changes in HOMAA.

NOTE

Once you enter a HOMADA value that is ≠ zero or ≠ HOMAD, S-curve profiling is enabled **only for homing move decelerations** (e.g., not for contouring decelerations, which require the PADA command). All subsequent homing moves for that axis must comply with this equation: 1/2 HOMAD ≤ HOMADA < HOMAD.

Increasing the HOMADA value above the pure S-curve level (HOMADA > 1/2 HOMAD), the time required to reach the target velocity and the target distance is decreased. However, increasing HOMADA also increases jerk.

The calculation for determining S-curve average accel and decel move times is as follows:

(A_{avg} = average accel or decel value):

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

Scaling (SCLA) affects HOMADA the same as it does for HOMAD.

*** For a more in-depth discussion on S-curve profiling, refer to the Custom Profiling chapter in the *6000 Series Programmer's Guide*.

Example:

```
SCALE0          ; Disable scaling
@MA0            ; Select incremental positioning mode
HOMA10,10       ; Set homing max. accel to 10 rev/sec/sec (axes 1 and 2)
HOMAA5,10       ; Set homing avg. accel to 5 rev/sec/sec on axis 1,
                ; and 10 rev/sec/sec on axis 2
HOMAD10,10      ; Set homing max. decel to 10 rev/sec/sec (axes 1 and 2)
HOMADA5,10      ; Set homing avg. decel to 5 rev/sec/sec on axis 1,
                ; and 10 rev/sec/sec on axis 2
HOM11XX         ; Execute negative-direction homing moves on axes 1 and 2.
                ; Axis 1 executes a pure S-curve.
                ; Axis 2 executes a trapezoidal profile.
```

HOMBAC Home Backup Enable

Type	Homing
Syntax	<!><@><a>HOMBAC<b><b><b><b>
Units	n/a
Range	b = 0 (disable), 1 (enable), or X (don't change)
Default	0
Response	HOMBAC: *HOMBAC0000 1HOMBAC: *1HOMBAC0
See Also	HOM, HOMA, HOMAA, HOMAD, HOMADA, HOMDF, HOMEDG, HOMLVL, HOMV, HOMVF, HOMZ

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

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

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

HOMDF Home Final Direction

Type	Homing
Syntax	<!><@><a>HOMDF<b><b><b><b>
Units	n/a
Range	b = 0 (positive-direction), 1 (negative-direction), or X (don't change)
Default	0
Response	HOMDF: *HOMDF0000 1HOMDF: *1HOMDF0
See Also	HOM, HOMA, HOMAA, HOMAD, HOMADA, HOMBAC, HOMEDG, HOMLVL, HOMV, HOMVF, HOMZ

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Home Final Direction (HOMDF) command specifies the direction the 6000 Series product is to be traveling when the home algorithm does its final approach. This command is operational when backup to home (HOMBAC) is enabled, or when homing to an encoder Z channel (HOMZ).

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

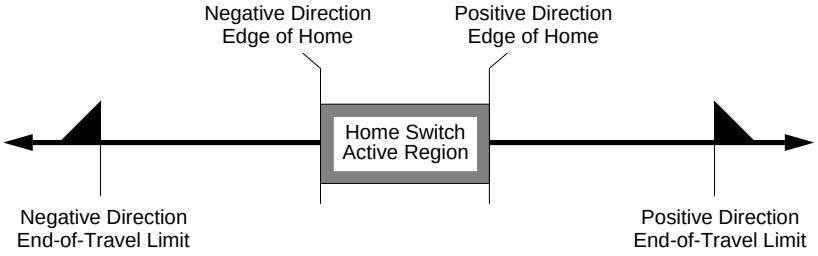
HOMEDG Home Reference Edge

Type	Homing
Syntax	<!><@><a>HOMEDG<b><b><b><b>
Units	n/a
Range	b = 0 (positive-direction edge), 1 (negative-direction edge), or X (don't change)
Default	0
Response	HOMEDG: *HOMEDG0000 1HOMEDG: *1HOMEDG0
See Also	HOM, HOMA, HOMAA, HOMAD, HOMADA, HOMBAC, HOMDF, HOMLVL, HOMV, HOMVF, HOMZ

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

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

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



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

HOMLVL Home Active Level

Type	Homing
Syntax	<!><@><a>HOMLVL<b><b><b><b>
Units	n/a
Range	b = 0 (active low), 1 (active high), or X (don't change)
Default	0
Response	HOMLVL: *HOMLVL0000 1HOMLVL: *1HOMLVL0
See Also	HOM, HOMA, HOMAA, HOMAD, HOMADA, HOMBAC, HOMDF, HOMEDG, HOMV, HOMVF, HOMZ, TLIM

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Home Active Level (HOMLVL) command defines the active state of the home input.

If a normally-open switch is used, the active level should be set to *active low* (HOMLVLØ). If a normally-closed switch is used, the active level should be set to *active high* (HOMLVL1).

If the device driving the Home input is off (not sinking current), the input will show (using the TLIM command) a zero (0) if the input has been defined as active low, and a one (1) if the input has been defined as active high. If the device driving the Home input is on (sinking current), the input will show a one (1) if the input has been defined as active low, and a zero (Ø) if the input has been defined as active high. The home input schematic is provided in the 6000 Series product's *Installation Guide.*

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

HOMV Home Velocity

		Product	Rev
Type	Homing	AT6n00	1.0
Syntax	<!><@><a>HOMV<r>,<r>,<r>,<r>	AT6n50	1.0
Units	r = units/sec (scalable with SCLV)	610n	4.0
Range	0.00000 - 1,600,000 (depends on scaling factor & PULSE)	615n	1.0
Default	1.0000	620n	1.0
Response	HOMV: *HOMV1.0000,1.0000,1.0000,1.0000 1HOMV: *1HOMV1.0000	625n	1.0
See Also	HOM, HOMA, HOMAA, HOMAD, HOMADA, HOMBAC, HOMDF, HOMEDG, HOMLVL, HOMVF, HOMZ, PULSE, SCALE, SCLV	6270	1.0

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

Steppers: The entered value is always in reference to motor steps, not encoder steps, regardless of the ENC command setting. If scaling is not enabled (SCALEØ), the velocity value is entered in motor revs/sec; this value is internally multiplied by the drive resolution (DRES) value to obtain a velocity value in motor steps/sec for the motor trajectory calculations. If scaling is enabled (SCALE1), the entered velocity value is internally multiplied by the velocity scaling factor (SCLV) to convert user units/sec to motor steps/sec.

Servos: If scaling is not enabled (SCALEØ), the velocity value is entered in encoder revs/sec, LDT inches/sec, or ANI volts/sec; encoder and LDT values are internally multiplied by the encoder resolution (ERES) value or the LDT resolution (LDTRES) value to obtain a velocity value in steps/sec for the motion trajectory calculations. If scaling is enabled (SCALE1), the entered velocity value is internally multiplied by the velocity scaling factor (SCLV) to convert user units/sec to encoder, LDT, or ANI steps/sec.

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

HOMVF Home Final Velocity

		Product	Rev
Type	Homing	AT6n00	1.0
Syntax	<!><@><a>HOMVF<r>,<r>,<r>,<r>	AT6n50	1.0
Units	r = units/sec (scalable with SCLV)	610n	4.0
Range	0.00000 - 1,600,000 (depends on scaling factor & PULSE)	615n	1.0
Default	0.1000	620n	1.0
Response	HOMVF: *HOMVF0.1000,0.1000,0.1000,0.1000 1HOMVF: *1HOMVF0.1000	625n	1.0
See Also	HOM, HOMA, HOMAA, HOMAD, HOMADA, HOMBAC, HOMDF, HOMEDG, HOMLVL, HOMV, HOMZ, PULSE, SCALE, SCLV	6270	1.0

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

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

Steppers: The entered value is always in reference to motor steps, not encoder steps, regardless of the ENC command setting. If scaling is not enabled (SCALEØ), the velocity value is entered in motor revs/sec; this value is internally multiplied by the drive resolution (DRES) value to obtain a velocity value in motor steps/sec for the motor trajectory calculations. If scaling is enabled (SCALE1), the entered velocity value is internally multiplied by the velocity scaling factor (SCLV) to convert user units/sec to motor steps/sec.

Servos: If scaling is not enabled (SCALEØ), the velocity value is entered in encoder revs/sec, LDT inches/sec, or ANI volts/sec; encoder and LDT values are internally multiplied by the encoder resolution (ERES) value or the LDT resolution (LDTRES) value to obtain a velocity value in steps/sec for the motion trajectory calculations. If scaling is enabled (SCALE1), the entered velocity value is internally multiplied by the velocity scaling factor (SCLV) to convert user units/sec to encoder, LDT, or ANI steps/sec.

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

HOMZ — Home to Encoder Z-channel Enable

Type	Homing
Syntax	<!><@><a>HOMZ<b><b><b><b>
Units	n/a
Range	b = 0 (disable), 1 (enable), or X (don't change)
Default	0
Response	HOMZ: *HOMZ0000 1HOMZ: *1HOMZ0
See Also	HOM, HOMA, HOMAA, HOMAD, HOMADA, HOMBAC, HOMDF, HOMEDG, HOMLVL, HOMV, HOMVF

Product	Rev
AT6n00	1.0
OEM-AT6n00	n/a
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

This command enables homing to an encoder z-channel after the initial home input has gone active. In stepper products, this function works in either motor step mode (ENCØ) or encoder step mode (ENC1). NOTE: The home limit input is required to go active prior to homing to the Z channel.

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

IF( ) — IF Statement

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

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

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

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

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

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

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

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

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

Multiple parentheses may not be used within the IF command.

Example:

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

[IN] Input Status

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

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Input Status (IN) command is used to assign the input value to a binary variable (3), or to make a comparison against a binary or hexadecimal value. To make a comparison against a binary value, the letter b (b or B) must be placed in front of the value. The binary value itself must only contain ones, zeros, or Xs (1, Ø, X, x). To make a comparison against a hexadecimal value, the letter h (h or H) must be placed in front of the value. The hexadecimal value itself must only contain the letters A through F, or the numbers Ø through 9.

Syntax: VARBn=IN where n is the binary variable number,
or [IN] can be used in an expression such as IF(IN=b11Ø1), or IF(IN=h7F)

The number of inputs available for assignment or comparison varies from one 6000 Series product to another; to ascertain the input bit assignments for your 6000 Series product refer to page 6. The function of the inputs is established with the INFNC command (although the [IN] command looks at all inputs regardless of their assigned function from the INFNC command). If it is desired to assign only one input value to a binary variable, instead of all the inputs, the bit select (.) operator can be used. For example, VARB1=IN.12 assigns input 12 to binary variable 1.

Example:

```
VARB1=IN           ; Input status assigned to binary variable 1
VARB2=IN.12        ; Input bit 12 assigned to binary variable 2
VARB2              ; Response if bit 12 is set to 1:
                   ; *VARB2=XXXX_XXXX_XXX1_XXXX_XXXX_XXXX_XXXX_XXXX
IF(IN=b111011X11)  ; If the input status contains 1's for inputs 1,2,3,5,6,8,& 9,
                   ; and 0 for input 4, do the commands following the IF statement
TREV               ; Transfer revision level
NIF                ; End IF statement
IF(IN=h7F00)       ; If the input status contains 1's for inputs 1,2,3,5,6,7,& 8,
                   ; and 0's for every other input, do the commands following the
                   ; IF statement
TREV               ; Transfer revision level
NIF                ; End IF statement
```

INDAX — Participating Axes

Type	Controller Configuration
Syntax	<!>INDAX<i>
Units	i = number axes to be controlled
Range	0 - 4 (Product dependent)
Default	Maximum number
Response	INDAX: *INDAX4
See Also	INDUST, LDTUPD, SSFR

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	n/a
615n	n/a
620n	1.0
625n	1.0
6270	1.0

The Participating Axes (INDAX) command defines the number of axes that will be controlled by the 6000 Series product. The default value includes all axes. This implies that all response commands will show a response for each axis.

If fewer axes are to be used, change the INDAX value. No report-backs or command parameters are accepted for axes excluded as a result of the INDAX command. For example, if you specify INDAX2 (use axes 1 and 2), the A command would show a response of *A1Ø.ØØØØ,1Ø,ØØØØ, and 4A command would show the response *INCORRECT AXIS.

By setting the number of participating axes less than the default, other items such as limits and stalls are not checked on the non-participating axes.

Servos: Changing the INDAX setting also changes the servo sampling frequency (refer to the SSFR command description for details).

Example:

```
INDAX3          ; Use only 3 axes
```

INDEB — Input Debounce Time

Type	Input
Syntax	<!>INDEB<i>,<i>
Units	1st i = input #; 2nd i = time in milliseconds (ms)
Range	1st i = 1-28 (product dependent); 2nd i = 2-250 (even #s)
Default	1st i = 1; 2nd i = 4 for general-purpose, 50 for stepper triggers and 24 for servo triggers
Response	INDEB: *INDEB1-24,4 *INDEB25,50 *INDEB26,50 *INDEB27,50 *INDEB28,50
See Also	INFNC, RE, REG, TIN, TRGFN

Product	Rev
AT6n00	2.1
AT6n50	1.0
610n	4.0
615n	1.0
620n	2.1
625n	1.1
6270	1.0

Using the Input Debounce Time (INDEB) command, you can change the input debounce time for all general-purpose inputs (one debounce time for all), or you can assign a unique debounce time to each of the trigger inputs. (Refer to page 6 for input bit assignments for your product.)

> **General-Purpose Input Debounce**: The input debounce time for the general-purpose inputs is the period of time that the input must be held in a certain state before the 6000 Series controller recognizes it. This directly affects the rate at which the inputs can change state and be recognized. The default debounce for general-purpose inputs is 4 milliseconds.
>
> **Trigger Input Debounce**: For trigger inputs, the 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 or registration move. The default debounce for trigger inputs on stepper products is 50 milliseconds, and for servo products it is 24 milliseconds. NOTE: When using the trigger input status as the condition for IF, ON, and WAIT statements, it is the non-debounced state that is recognized; therefore, rapid transitions, as short as one system update (steppers: 2 ms for steppers; servos: depends on INDAX and SSFR settings), will be noticed by these statements.

The INDEB command syntax is INDEB<i>,<i>. The first <i> is the input number and the second <i> is the debounce time **in even increments** of milliseconds (ms). The debounce time range is 2 - 250 ms. Input bit patterns vary by product — to ascertain the pattern for your product, refer to page 6 of this document.

Example (for the AT6400, and other products with 24 general-purpose inputs and at least 2 trigger inputs):

```
INDEB5,6        ; Assign inputs 1 through 24 (all general-purpose inputs)
                ; a debounce time of 6 ms
INDEB25,10      ; Assign Trigger A (input 25) a debounce time of 10 ms
INDEB26,12      ; Assign Trigger B (input 26) a debounce time of 12 ms
```

INDUSE — Enable/Disable User Status

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

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

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

Example:

```
INDUSE1         ; Enable user status
```

INDUST User Status Definition

		Product	Rev
Type	Controller Configuration	AT6n00	1.0
Syntax	<!>INDUST<i><-<i><c>>	AT6n50	1.0
Units	See description below	610n	4.0
Range	1st i = 1 - 16; 2nd i = 1 - 32; c = A, B, C, D, I, J, K, L, M, N or O	615n	1.0
Default	See description below	620n	1.0
Response	INDUST: *INDUST1-1A AXIS 1 STATUS - STATUS OFF (...repeated for all 16 user status bits...) *INDUST16-4D AXIS 4 STATUS - STATUS OFF INDUST1: *INDUST1-1A AXIS 1 STATUS - STATUS OFF	625n	1.0
		6270	1.0
See Also	[AS], [ASX], [IN], INDUSE, ONUS, [SS], TAS, TASX, TIN, TINT, TSS, TUS, [US]		

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

AT6400:

1-4 correspond to the first 4 bits of the axis status (AS) for axis 1
5-8 correspond to the first 4 bits of the axis status (AS) for axis 2
9-12 correspond to the first 4 bits of the axis status (AS) for axis 3
13-16 correspond to the first 4 bits of the axis status (AS) for axis 4

AT6200, AT6n50, 620n, 625n, and 6270:

1-8 correspond to the first 8 bits of the axis status (AS) for axis 1
9-16 correspond to the first 8 bits of the axis status (AS) for axis 2

610n and 615n:

1-16 correspond to the first 16 bits of the axis status (AS)

The purpose of this command is to allow the user to create his or her own meaningful status word. It allows the user to place certain status information in the order they prefer.

The syntax for INDUST<i><-<i><c>> is described as follows:

First <i> corresponds to the user status bit being defined (16 maximum).

Second <i> corresponds to the bit of the system status (SS), the axis status (AS), the input status (IN), the interrupt status (TINT), or the extended axis status (ASX).

The <c> defines what status to use:

<c> values	Function
A	Use axis status (AS) for axis 1
B	Use axis status (AS) for axis 2
C	Use axis status (AS) for axis 3 (AT6400 and AT6450)
D	Use axis status (AS) for axis 4 (AT6400 and AT6450)
E, F, G, H	Reserved
I	Use system status (SS)
J	Use input status (IN)
K	Use interrupt status (TINT) — AT6400 & AT6n50
L	Use extended axis status (ASX) for axis 1
M	Use extended axis status (ASX) for axis 2
N	Use extended axis status (ASX) for axis 3
O	Use extended axis status (ASX) for axis 4

Example

```
INDUSE1          ; Enable user status
INDUST1-5A       ; User status bit 1 defined as axis 1 status bit 5
INDUST2-3D       ; User status bit 2 defined as axis 4 status bit 3
INDUST3-5J       ; User status bit 3 defined as input 5
INDUST4-1K       ; User status bit 4 defined as interrupt status bit 1
INDUST16-2I      ; User status bit 16 defined as system status bit 2
```

INEN — Input Enable

		Product	Rev
Type	Input or Program Debug Tool		
Syntax	<!>INEN<d><d>...<d> (one <d> for each input)	AT6n00	1.0
Units	n/a	AT6n50	1.0
Range	d = 0 (disable, leave off), 1 (disable, leave on), E (enable), or X (don't change)	610n	4.0
Default	E	615n	1.0
Response	INEN: *INENEEEE_EEEE_EEEE_EEEE_EEEE_EEEE_EEEE	620n	1.0
See Also	[IN], INFEN, INFNC, INLVL, INPLC, INSTW, TIN, TSTAT	625n	1.0
		6270	1.0

The Input Enable (INEN) command enables or disables specific inputs. The default state for each input is the enabled condition. When an input is enabled, the function programmed for that input (INFNC), will be active. **The INEN command has no effect on trigger inputs when they are configured as trigger interrupt inputs with the INFNCi-H command.**

The inputs can be disabled and set to a specific level (ON or OFF) through the use of the INEN command. For instance, INEN1 disables input 1 but leaves it in the ON state (the TIN command will show input 1 as active). INENØ disables input 1 but leaves it in the OFF (inactive) state. To re-enable input 1, issue the INENE command.

Input bit assignments for the INEN command vary by product. The input bit patterns for all 6000 products are provided page 6 of this document. The inputs are numbered 1 to *n* (*n* depends on the product) from left to right.

By disabling the inputs and setting them to a specific level, input simulation can be accomplished without wiring the inputs. **You cannot simulate an encoder capture or registration input with the INEN command.**

Example:
```
DEF tester        ; Begin definition of program tester
WHILE(IN=b11X10)  ; While inputs 1,2, and 4 are active, and input 5 is not
                  ; active, execute the statements between the WHILE and NWHILE
GO1100            ; Initiate motion on axes 1 and 2
NWHILE            ; End WHILE statement
END               ; End definition of program tester
INEN11X10         ; Disable inputs 1,2,4, and 5, and set inputs 1, 2 and 4 in
                  ; the active state, and input 5 in the inactive state
RUNtester         ; Initiate program tester
!INEN00000        ; Disable inputs 1,2,3,4, and 5, and leave them in the
                  ; inactive state
INENeeeee         ; Re-enable inputs 1 through 5
```

INFEN — Input Function Enable/Disable

		Product	Rev
Type	Input		
Syntax	<!>INFEN<b>	AT6n00	1.0
Units	n/a	AT6n50	1.0
Range	b = 0 (disable) or 1 (enable)	610n	4.0
Default	0 (1 for 610n, 615n and 6201 only)	615n	1.0
Response	INFEN: *INFEN0	620n	1.0
See Also	DRFLVL, INFNC, RE, REG, TAS, TIN	625n	1.0
		6270	1.0

The Input Function Enable/Disable (INFEN) command enables the drive fault input and input functions (INFNC). If this command is not enabled, the drive fault input will not indicate a drive fault in the TAS command (OEM-AT6400: the drive fault input is not available). **Input functions defined with the INFNC command will have no effect unless INFEN is enabled.**

NOTE

Before you enable this command, verify that the drive fault level (DRFLVL) is set correctly for each axis.

Example:
```
INFEN1          ; Enable input functions
```

INFNC Input Function

		Product	Rev
Type	Input	AT6n00	1.4
Syntax	<!>INFNC<i>-<<a>c>	AT6n50	1.0
Units	i = input #, a = axis #, c = function identifier letter	610n	4.0
Range	i = 1 - 28 (product dependent); a = 1 - 4 (product dependent); c = A - P	615n	1.0
Default	A	620n	1.5
Response	INFNC: *INFNC1-A NO FUNCTION INPUT - STATUS OFF (...repeated for all inputs...) *INFNC28-A NO FUNCTION INPUT - STATUS OFF INFNC1: *INFNC1-A NO FUNCTION INPUT - STATUS OFF	625n	1.0
		6270	1.0
See Also	COMEXR, COMEXS, [ER], ERROR, [IN], INDAX, INDEB, INEN, INFEN, INLVL, INPLC, INSELP, INSTW, K, KDRIVE, PSET, [SS], SSFR, TER, TIN, TRGFN, TSS, TSTAT		

The Input Function (INFNC) command defines the function of each individual input, where *i* is the input bit number, *a* is an axis number if required, or the program number for the case of input function P, and *c* is the function. The number of inputs and axes differ from one 6000 Series product to another. All function definitions given below will specify whether an axis number is required.

ENABLE THE INPUT FUNCTIONS

The INFEN1 command must be issued before you can use the input functions defined with the INFNC command, except INFNCi-A.

Using the Input Debounce Time (INDEB) command, you can change the input debounce time for all general-purpose inputs (one debounce time for all), or you can assign a unique debounce time to each of the trigger inputs. Refer to the INDEB command description for details.

Input bit assignments vary by product. The input bit patterns for all 6000 products is provided on page 6.

Input Scan Rate: The programmable inputs are scanned once per *system update* (steppers: 2 milliseconds; servos: depends on current INDAX and SSFR settings — see table in SSFR command description).

Identifier **Function Description**

A **No special function** - Normal input, used with the IN assignment

B **BCD Program Select** - BCD input assignment to programs, lowest numbered input is least significant bit (LSB). BCD values for inputs are as follows:

	BCD Value
Least Significant Bit Value	1
.	2
.	4
.	8
.	10
.	20
.	40
.	80
Most Significant Bit Value	100

Note: If fewer inputs than shown above are defined to be Program Select Inputs, then the highest input number defined as a Program Select Input is the most significant bit.

An input defined as a BCD Program Select Input will not function until the INSELP command has been enabled. The trigger inputs (**TRG-A** through **TRG-D**—typically assigned to inputs 25 through 28, but varies by product) cannot be used for this function.

C **Kill** - Kills motion on all axes and halts all command processing (refer to K and KDRIVE command descriptions for further details on the *kill* function). This is an edge detection function and is not intended to inhibit motion. To inhibit motion, use the Pause/Resume function (INFNCi-E). When enabled with the ERROR command, bit #6 of the TER and ER commands will report the kill status.

Identifier	Function Description

D **Stop** - Stops motion. Axis number is optional; if no axis number is specified, motion is stopped on all axes. If COMEXS is set to zero (COMEXSØ), program execution will be terminated. If COMEXS is set to 1 (COMEXS1), command processing will continue. With COMEXS set to 2 (COMEXS2), program execution is terminated, but the INSELP value is retained. Motion deceleration during the stop is controlled by the AD & ADA commands. If error bit #8 is enabled (e.g., ERROR.8-1), activating a Stop input will set the error bit and cause a branch to the ERRORP program.

E **Pause/Continue** - If COMEXR is disabled (COMEXRØ), then only command execution pauses, not motion. With COMEXR enabled (COMEXR1), both command and motion execution are paused. After motion stops, you can release the input or issue a continue (!C) command to resume command processing.

F **User Fault** - Refer to the ERROR command. If error bit #7 is enabled (e.g., ERROR.7-1), activating a User Fault input will set the error bit and cause a branch to the ERRORP program. **CAUTION**: Activating the user fault input sends an !K command to the controller, "killing" motion on all axes (refer to the K command description for ramifications).

G **Reserved**

H **Trigger Interrupt (Position Capture and Registration)** - Only the trigger inputs (**TRG-A** through **TRG-D** – input numbers vary by product) can be used as interrupt inputs. The axis number is not required; if an axis number is specified, it will be ignored. You can change the debounce time of each trigger input with the INDEB command (see INDEB command description for details). If you issue a PSET command, the captured positions will be offset by the specified PSET command value. The INFNCi-H command also enables the specified trigger to be used for TRGFN functions.

Steppers: Activating any trigger input defined as a *trigger interrupt* input will capture the position of all the encoders and motors on all axes (within 50µs of input activation). If registration is enabled (with the RE command), defined registration move(s) will occur based on the captured positions and which trigger input is activated. Use the TPCE and TPCM commands to read the captured encoder and motor positions. You can also use the PCE and PCM commands to assign or compare the captured encoder and motor positions. (The OEM-AT6n00 does not support the use of encoders and, therefore, cannot use the TPCE and PCE commands.)

Servos: When a *trigger interrupt* input is activated, the commanded position and the positions of all feedback devices on all axes are captured at one time. The position information is stored in registers and is available through the use of transfer and assignment/comparison commands (see table below).

Captured Information	Transfer	Assignment/Comparison
Commanded position	TPCC	PCC
LDT position	TPCL	PCL
Encoder or resolver position	TPCE or TCA	PCE or CA
ANI value (-ANI option)	TPCA	PCA

If you are capturing the position/value of an encoder, resolver, LDT or ANI when it is selected as the feedback source with the SFB command, the captured position is interpolated from the last sampled position and velocity of the feedback device, and the time elapsed since the last sample. *The position sample rate is determined by the SSFR and INDAX commands* (see *System Update* column in SSFR table). *The accuracy of the position capture is ±50µs x velocity.*

If you are capturing the position of the encoder, resolver, LDT or ANI when it is NOT selected with the SFB command, the last sampled position is simply stored as the captured position. Therefore, the accuracy is one system update period (determined by the SSFR and INDAX commands).

Identifier	Function Description

H (con't.) Regardless of the SFB selection, one encoder position is latched in hardware within ±1 encoder count (at max. encoder frequency) when its dedicated trigger input is activated (see table below).

Encoder	AT6250	AT6450	615n*	625n	6270	OEM625n
ENCODER 1	TRG-A	TRG-A	TRG-A	TRG-A	TRG-A	TRG-A
ENCODER 2	TRG-B	TRG-B	TRG-B	TRG-B	n/a	TRG-B
ENCODER 3	TRG-C	TRG-C	n/a	TRG-C	n/a	n/a
ENCODER 4	n/a	TRG-D	n/a	n/a	n/a	n/a

** 615n: TRG-A captures the internal resolver and TRG-B captures the external encoder.*

The captured commanded position is always interpolated from the last sampled position (of the feedback device selected with the SFB command) and position error, and the time elapsed since the last sample.

Registration is not available in the servo controllers.

I **Interrupt to PC-AT** - Will cause the bus-based 6000 controller to interrupt the PC-AT. This is not a valid input function for the stand-alone products.

J **JOG positive-direction** - Will jog the axis specified in a positive-direction. The JOG command must be enabled for this function to work. **Axis number required**.

K **JOG negative-direction** - Will jog the axis specified in a negative-direction. The JOG command must be enabled for this function to work. **Axis number required**.

L **JOG Speed Select** - Selects the high or low velocity range while jogging. If the input is active, the high jog velocity range will be selected. Axis number is optional. If no axis number is designated, it defaults to all axes.

M, N, O **Reserved**

P **Program Select** - One to one correspondence for input vs. program number. The program number comes from the TDIR command. The number specified before the program name is the number to specify within this input definition (e.g., INFNC1-3P, where 3 equals the program number [TDIR]). An input defined as a Program Select Input will not function until the INSELP command has been enabled. The trigger inputs (**TRG-A** through **TRG-D**) cannot be used for this function.

Q **Program Security** - Issuing the INFNCi-Q command enables the *Program Security* feature and assigns the *Program Access* function to the specified programmable input.

The program security feature denies you access to the DEF, DEL, ERASE, MEMORY, and INFNC commands until you activate the program access input. Being denied access to these commands effectively restricts altering the user memory allocation. If you try to use these commands when program security is active (program access input is not activated), you will receive the error message *ACCESS DENIED. *The INFNCi-Q command is not saved in battery-backed RAM, so you may want to put it in the start-up program (STARTP).*

For example, once you issue the INFNC22-Q command, input #22 is assigned the program access function and access to the DEF, DEL, ERASE, MEMORY, and INFNC commands will be denied until you activate input #22.

To regain access to these commands without the use of the program access input, you must issue the INFENØ command to disable programmable input functions, make the required user memory changes, and then issue the INFEN1 command to re-enable the programmable input functions.

Example:

```
INFEN1          ; Enable input functions
INFNC1-D        ; Input #1 defined to be a stop input for all axes
```

INLVL — Input Active Level

Type	Input
Syntax	<!>INLVL<b><b>...<b><b>
Units	n/a
Range	b = 0 (active low), 1 (active high), or X (don't change)
Default	0
Response	INLVL: *INLVL0000_0000_0000_0000_0000_0000_0000_0000
See Also	INEN, INFEN, INFNC, INPLC, INSTW

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Input Active Level (INLVL) command defines the active state of all programmable inputs. To determine the input bit assignments for your 6000 Series product, refer to page 6 of this document.

If the device driving the input is off (not sinking current), the input will show (using the TIN command) a zero (0) if the input has been defined as active low, and a one (1) if the input has been defined as active high. If the device driving the input is on (sinking current), the input will show a one (1) if the input has been defined as active low, and zero (0) if the input has been defined as active high. The default state is active low (INLVL0). The input schematics are provided in each 6000 Series product's *Installation Guide*.

Example:

```
INLVL0101       ; Set active level: inputs 1 & 3 active low, 2 & 4 active high
```

[INO] — Other Input Status

Type	Assignment or Comparison
Syntax	See below
Units	n/a
Range	n/a
Default	n/a
Response	n/a
See Also	[IN], JOY, [LIM], TINO, TINOF

Product	Rev
AT6n00	1.0
OEM-AT6n00	n/a
AT6n50	1.0
610n	4.0
615n	4.0
620n	1.0
625n	1.0
6270	1.0

The Other Input Status (INO) command is used to assign an other input value to a binary variable, or to make a comparison against a binary or hexadecimal value. To make a comparison against a binary value, the letter b (b or B) must be placed in front of the value. The binary value itself must only contain ones, zeros, or Xs (1, Ø, X, x). To make a comparison against a hexadecimal value, the letter h (h or H) must be placed in front of the value. The hexadecimal value itself must only contain the letters A through F, or the numbers Ø through 9.

Syntax: VARBn=INO where n is the binary variable number
or [INO] can be used in an expression such as IF(INO=b11Ø1), or IF(INO=hØ2)

There are 8 other inputs available for assignment or comparison. If it is desired to assign only one bit (one specific input) value to a binary variable, instead of all 8, use the bit select (.) operator . For example, VARB1=INO.2 assigns other input 2 to binary variable 1.

Format for binary assignment: bbbbbbbb
^ Bit #1 ^ Bit #8

Bit	**Function** (1 = yes, 0 = no)	Location
1	Joystick Auxiliary Input Active	Joystick Connector Pin 19
2	Joystick Trigger Input Active	Joystick Connector Pin 18
3	Joystick Axes Select Input Active	Joystick Connector Pin 15
4	Joystick Velocity Select Input High	Joystick Connector Pin 16
5	Joystick Release Input Active	Joystick Connector Pin 17
6	Pulse Cutoff Input (steppers only) Enable input (servos only) (1 = OK for motion)	P-CUT terminal on AUX connector ENBL terminal on the AUX connector
7	Not used, always 0	----------
8	Not used, always 0	----------

Example:

```
VARB1=INO          ; Other input status assigned to binary variable 1
VARB2=INO.4        ; Other input bit 4 assigned to binary variable 2
VARB2              ; Response if bit 4 is set to 1: *VARB2=XXX1_XXXX
IF(INO=b111011X)   ; If the other input status contains 1's for inputs 1, 2, 3,
                   ; 5, and 6, and a 0 for input 4, do the commands following the
                   ; IF statement until the NIF statement
TREV               ; Transfer revision level
NIF                ; End if statement
IF(INO=h77)        ; If the other input status contains 1's for inputs 1, 2, 3,
                   ; 5, 6, and 7, and 0 for input 4 and 8, do the commands
                   ; following the IF statement until the NIF statement
TREV               ; Transfer revision level
NIF                ; End if statement
```

INPLC — Establish PLC Data Inputs

Type	Input
Syntax	<!>INPLC<i>,<i-i>,<i>,<i>
Units	See below
Range	See below
Default	1,0–0,0,0
Response	INPLC1: *INPLC1,0-0,0,0
See Also	INEN, INFNC, INLVL, INSTW, OUTPLC, [TW]

Product	Rev
AT6n00	1.0
OEM-AT6n00	n/a
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Establish PLC Data Inputs (INPLC) command, in combination with the OUTPLC command, configure the inputs and outputs to read data from a parallel I/O device such as a PLC (Programmable Logic Controller), or a passive thumbwheel module. The actual data transfer occurs with the TW command. Refer to the TW command for a description of the data transfer process.

The INPLC command has four fields (<i>,<i-i>,<i>,<i>):

Data Field	Description
Field 1: <i>	**Set #**: There are 4 possible INPLC sets (1-4). This field identifies which set to use.
Field 2: <i-i>	**Input #s**: Data is read into the 6000 Series product through the programmable inputs. This field identifies the indexer inputs to be used with the TW command. The first number is the first input, and the second number is the last input. The inputs must be consecutive. The number of inputs should be 8, because two BCD digits are read per data strobe.
Field 3: <i>	**Sign Input #**: This field identifies which input is designated to provide sign information. A zero specified in the command field specifies no sign information. An active signal on the input designated as the sign input indicates a negative data entry.
Field 4: <i>	**Data Valid Input #**: This field identifies which input is designated to be the data valid handshake input. A zero in this field indicates that there will be no data valid handshake input used. When an input is specified as a data valid, the input must be active in order for data to be read. If the input is not active, data will not be read until the signal becomes active.

To disable a specific PLC set, enter INPLCn,Ø-Ø,Ø,Ø where n is the PLC set (1-4).

Example:

```
INPLC2,1-8,9,10   ; Set INPLC set 2 as BCD digits on inputs 1 - 8, with input 9
                  ; as the sign bit, and input 10 as the data valid
OUTPLC2,1-4,5,50  ; Set OUTPLC set 2 as output strobes on outputs 1 - 4, with
                  ; output 5 as the output enable bit, and strobe time of 50
                  ; milliseconds
A(TW6)            ; Read data into axis 1 acceleration using INPLC set 2 and
                  ; OUTPLC set 2 as the data configuration
```

INSELP — Select Program Enable

Type	Input
Syntax	<!>INSELP<i>,<i>
Units	See below
Range	1st i = 0, 1, or 2; 2nd i = 0 - 5000
Default	0,0
Response	INSELP: *INSELP0,0
See Also	COMEXS, INEN, INFEN, INFNC, INLVL, INPLC, INSTW, [SS], TDIR, TSS

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Select Program Enable (INSELP) command enables program selection by inputs. In addition, the command establishes the strobe time for the inputs, and if programs are selected on a one-to-one basis (INFNCi-iP) or on a BCD basis (INFNCi-B). When programs are selected on a one-to-one basis, each input defined with the INFNCi-iP command will run a specific program upon activation. When programs are selected by BCD values, each input defined by the INFNCi-B command will contribute to the BCD value, which corresponds to the program number. The program number is derived from the order in which the programs were defined (DEF). The first program defined is program #1, the second defined is program #2, etc. To verify which program number corresponds to each program, use the TDIR command. The number in front of the program name is the program number.

First i = Enable or disable function (Ø = Disable, 1 and 2 = Enable). Use INFNCi-B inputs if i = 1, or INFNCi-iP inputs if i = 2, to select program..

Second i = Strobe Time in milliseconds for inputs used to select program. The input must be active at the end of the strobe time for it to be recognized as a valid selection. The inputs are scanned once per *system update* (steppers: 2 milliseconds; servos: depends on current INDAX and SSFR settings — see table in SSFR command description).

The Kill (!K) command releases this mode, in addition to INSELPØ. The Stop (!S) command or an input defined as a stop input will also release this mode, as long as COMEXS has been disabled.

Example:

```
INFNC1-1P       ; Input #1 defined to select program #1
INFNC2-2P       ; Input #2 defined to select program #2
INFNC3-7P       ; Input #3 defined to select program #7
INSELP2,50      ; Enable continuous scan of inputs to select a program to run
```

INSTW — Establish Thumbwheel Data Inputs

Type	Input
Syntax	<!>INSTW<i>,<i-i>,<i>
Units	See below
Range	See below
Default	1,0—0,0
Response	INSTW1: *INSTW1,0-0,0
See Also	INEN, INFNC, INLVL, INPLC, OUTTW, [SS], TSS, [TW]

Product	Rev
AT6n00	1.0
OEM-AT6n00	3.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Establish Thumbwheel Data Inputs (INSTW) command, in combination with the OUTTW command, configure the inputs and outputs to read data from an active thumbwheel device such as Compumotor's TM8 Thumbwheel Module. The actual data transfer occurs with the TW command. Refer to the TW command for a description of the data transfer process.

The INSTW command has three fields (<i>,<i-i>,<i>):

Data Field	Description
Field 1: <i>	**Set #**: There are 4 possible INSTW sets (1-4). This field identifies which set to use.
Field 2: <i-i>	**Input #s**: Data is read into the 6000 Series product through the programmable inputs. This field identifies the indexer inputs to be used with the TW command. The first number is the first input, and the second number is the last input. The inputs must be consecutive. The number of inputs should be compatible to the thumbwheel device (4 for the TM8 module).

Field 3: <i> **Sign Input #**: This field identifies which input is designated to provide sign information. A zero specified in the command field specifies no sign information. An active signal on the input designated as the sign input indicates a negative data entry.

To disable a specific thumbwheel set, enter INSTWn,Ø-Ø,Ø where n is the thumbwheel set (1-4).

Example:
```
INSTW2,1-4,5    ; Set INSTW set 2 as BCD digits on inputs 1 - 4, with input 5 as
                ; the sign bit
OUTTW2,1-3,4,50 ; Set OUTTW set 2 as output strobes on outputs 1 - 3, with output
                ; 4 as the output enable bit, and strobe time of 50 milliseconds
A(TW2)          ; Read data into axis 1 acceleration using INSTW set 2
                ; and OUTTW set 2 as the data configuration
```

INTCLR Clear Interrupt Condition Status

Type	Interrupt to PC-AT
Syntax	<!>INTCLR<.i>
Units	i = interrupt status bit #
Range	i = 1 - 32
Default	n/a
Response	INTCLR: No response, instead INTCLR clears all interrupt status bits
See Also	INTHW, INTSW, TINT

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	n/a
615n	n/a
620n	n/a
625n	n/a
6270	n/a

This command clears the interrupt status registers and the TINT status command of any interrupt condition flags that may have occurred. When using the fast status registers, this command is required to clear the interrupt status, because that read does not automatically clear the active interrupt status bits.

If only one interrupt is to be cleared, use the bit select (.) and the corresponding bit number. For example, to clear interrupt bit number 13, type in INTCLR.13.

Example:
```
INTCLR          ; Clear all interrupt status bits
INTCLR.12       ; Clear interrupt status bit 12
```

INTHW Hardware Interrupt Enable

Type	Interrupt to PC-AT
Syntax	<!>INTHW<b><b>...<b><b> (one b for each of 32 interrupts)
Units	n/a
Range	b = 0 (disable), 1 (enable), or X (don't change)
Default	0
Response	INTHW: *INTHW0000_0000_0000_0000_0000_0000_0000_0000
See Also	INTSW, TINT

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	n/a
615n	n/a
620n	n/a
625n	n/a
6270	n/a

The Hardware Interrupt Enable (INTHW) command determines which interrupt conditions will cause an interrupt to the PC-AT hardware. There are 30 interrupt conditions that can cause an interrupt. There is no limit to the number of interrupt conditions that may be enabled, all 30 if desired. The order of the interrupt conditions is given below. The bits are defined from left to right, 1 to 32.

Format for binary assignment:
```
bbbb_bbbb_bbbb_bbbb_bbbb_bbbb_bbbb_bbbb
^                                     ^
Bit #1                                Bit #32
Location                              Location
```

To enable a specific interrupt, place a 1 in the corresponding bit location (b) in the INTHWbb....bbb command. To disable a specific interrupt bit, place a Ø in the corresponding bit location.

NOTE: A specific interrupt bit can also be enabled by specifying the bit and the state of the bit (Ø=Disable, 1 = Enable). For example, the command INTHW.29-1 enables bit 29, whereas INTHW.29-Ø disables bit 29.

Bit #	Function
1	Software Interrupt #1 (See INTSW)
2	Software Interrupt #2
3	Software Interrupt #3
4	Software Interrupt #4
5	Software Interrupt #5
6	Software Interrupt #6
7	Software Interrupt #7
8	Software Interrupt #8
9	Software Interrupt #9
10	Software Interrupt #10
11	Software Interrupt #11
12	Software Interrupt #12
13	Software Interrupt #13
14	Software Interrupt #14
15	Software Interrupt #15
16	Software Interrupt #16

Bit #	Function
17	Command Buffer Full
18*	Pulse Cutoff (steppers) or Enable (servos) Activated
19	Program Complete
20*	Drive Fault on any Axis
21	Reserved
22	Reserved
23	Limit Hit - hard or soft limit, on any axis
24*	Stall Detected (steppers) or Position Error (servos) on any axis
25	Timer (TIMINT)
26*	Counter (CNTINT) – steppers only
27	Input - any of the inputs defined by INFNCi-I
28	Command Error
29	Motion Complete on Axis 1
30	Motion Complete on Axis 2
31	Motion Complete on Axis 3 (AT6400 & AT6450)
32	Motion Complete on Axis 4 (AT6400 & AT6450)

* Not applicable to the OEM-AT6n00

The interrupt that is generated will interrupt the PC-AT on one of eight separate interrupt request lines, IRQ3, IRQ4, IRQ5, IRQ7, IRQ10, IRQ11, IRQ12, or IRQ15. The interrupt request line (IRQ) to be interrupted is determined by a bank of 8 DIP switches on the bus-based 6000 Series controller card. DIP switch #1 on selects IRQ3, DIP switch #2 on selects IRQ4, etc.

INTSW — Force Software Interrupt

Type	Interrupt to PC-AT
Syntax	<!>INTSW<i>
Units	i = software interrupt #
Range	i = 1 - 16
Default	n/a
Response	n/a
See Also	INTCLR, INTHW, TINT

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	n/a
615n	n/a
620n	n/a
625n	n/a
6270	n/a

This command forces a specific software interrupt. Sixteen different software interrupts are available. By forcing an interrupt condition, the user can customize the program to generate specific software interrupts at predefined places in his or her program.

In order for the software interrupt to interrupt the PC, that specific software interrupt must also be enabled. The interrupt is enabled with the INTHW command.

The PC software must determine the cause of the interrupt. This is accomplished by polling the controller's interrupt status register (interrupt status registers are in fast status register block) for the interrupt information. Once the interrupt status register has been read, the interrupt conditions must be cleared with the INTCLR command. For more information on the fast status registers, refer to the Fast Status Register section in the *6000 Series Programmer's Guide*. The interrupt information can also be obtained from the TINT command. Once the TINT command is entered, the interrupt status bits are cleared.

Example:

```
INTHW1          ; Enable software interrupt #1
A20,20          ; Set acceleration to 20 units/sec/sec on axes 1 and 2
V2,2            ; Set velocity to 2 units/sec on axes 1 and 2
D25000,25000    ; Set move distance to 25000 units on axes 1 and 2
GO11            ; Initiate motion on axes 1 and 2
INTSW1          ; Force software interrupt 1 as soon as the moves on axes 1 and
                ; 2 are finished.
; ************************************************************************
; * Note: After the interrupt occurs, it is the application program's     *
; * responsibility to examine the 6000 product's interrupt status register *
; * in the fast status area, or issue the TINT command to determine the   *
; * cause of the interrupt.                                              *
; ************************************************************************
```

JOG — Jog Mode Enable

Type	Jog
Syntax	<!><@><a>JOG<b><b><b><b>
Units	n/a
Range	b = 0 (disable), 1 (enable), or X (don't change)
Default	0
Response	JOG: *JOG0000 1JOG: *1JOG0
See Also	DJOG, JOGA, JOGAA, JOGAD, JOGADA, JOGVH, JOGVL, INFEN, INFNC

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

This command enables jog mode on the appropriate axis. Once jog mode has been enabled, the jog inputs can be used to produce motion on the specific axis. The inputs that will be used as jog inputs are determined by the INFNC command. Once the jog inputs have been enabled, they will remain enabled, and able to jog at any time while the motor is *in position*. Or in other words, as long as the motor is not moving the jog inputs will be active.

After processing the JOG1 command, command processing does not stop and wait for the jog mode to be disabled (JOGØ). Instead, the jog inputs are enabled and command processing continues with the first command after the JOG1 command.

WARNING
If a jog input is active when jog mode is enabled, motion will occur.

To disable jog mode, issue the JOGØ command (to the appropriate axis) at any point in the program.

NOTE: If you are using an RP240 operator panel, you can enable the RP240 Jog Mode with the DJOG1 command and use the RP240's arrow keys to jog individual axes. To disable the RP240 Jog Mode, use the !DJOGØ command or press the RP240's **MENU RECALL** button.

Example:

```
SCLA25000,25000  ; Set accel scaling factor on axes 1 & 2 to 25000 steps/unit/unit
SCLV25000,25000  ; Set velocity scaling factor on axes 1 and 2 to 25000 steps/unit
SCALE1           ; Enable scaling
INFEN1           ; Enable Input Functions
INFNC1-L         ; Input #1 defined as jog velocity select input
INFNC2-1J        ; Input #2 defined as jog positive-direction input for axis #1
INFNC3-1K        ; Input #3 defined as jog negative-direction input for axis #1
INFNC4-2J        ; Input #4 defined as jog positive-direction input for axis #2
INFNC5-2K        ; Input #5 defined as jog negative-direction input for axis #2
JOGA100,100      ; Jog acceleration set to 100 units/sec/sec on both axes
JOGAD200,200     ; Jog deceleration set to 200 units/sec/sec on both axes
JOGVH10,8        ; The velocity when the jog velocity select input is high is
                 ; 10 units/sec on axis #1 and 8 units/sec on axis 2
JOGVL1,.8        ; The velocity when the jog velocity select input is low is
                 ; 1 units/sec on axis #1 and 0.8 units/sec on axis 2
JOG1100          ; Enable jog mode on axes 1 and 2. When an input occurs on
                 ; input 2, input 3, input 4, or input 5, the motor will move at
                 ; the appropriate jog velocity until the input is released
WAIT(IN=bXXXXX1) ; Wait for input #6 to become active. Input #6 is being used
                 ; as a signal to disable jog mode.
JOG0000          ; Disable jog mode on all axes
```

JOGA — Jog Acceleration

		Product	Rev
Type	Jog		
Syntax	<!><@><a>JOGA<r>,<r>,<r>,<r>	AT6n00	1.0
Units	r = units/sec/sec	AT6n50	1.0
Range	0.00025 - 24,999,999 (depending on the scaling factor)	610n	4.0
Default	10.0000	615n	1.0
Response	JOGA: *JOGA10.0000,10.0000,10.0000,10.0000 1JOGA: *1JOGA10.0000	620n	1.0
		625n	1.0
See Also	DJOG, JOG, JOGAA, JOGAD, JOGADA, JOGVH, JOGVL, INFNC, SCALE, SCLA	6270	1.0

The Jog Acceleration (JOGA) command specifies the acceleration to be used upon receiving a jog input.

Steppers: The entered value is always in reference to motor steps, not encoder steps, regardless of the ENC command setting. If scaling is not enabled (SCALEØ), the acceleration value is entered in motor revs/sec/sec; this value is internally multiplied by the drive resolution (DRES) value to obtain an acceleration value in motor steps/sec/sec for the motion trajectory calculations. If scaling is enabled (SCALE1), the entered acceleration value is internally multiplied by the acceleration scaling factor (SCLA) to convert user units/sec/sec to motor steps/sec/sec.

Servos: If scaling is not enabled (SCALEØ), the acceleration value is entered in encoder or resolver revs/sec/sec, LDT inches/sec/sec, or ANI volts/sec/sec; encoder or resolver and LDT values are internally multiplied by the encoder resolution (ERES) value or the LDT resolution (LDTRES) value to obtain an acceleration value in steps/sec/sec for the motion trajectory calculations. If scaling is enabled (SCALE1), the entered acceleration value is internally multiplied by the acceleration scaling factor (SCLA) to convert user units/sec/sec to encoder, resolver, LDT, or ANI steps/sec/sec.

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

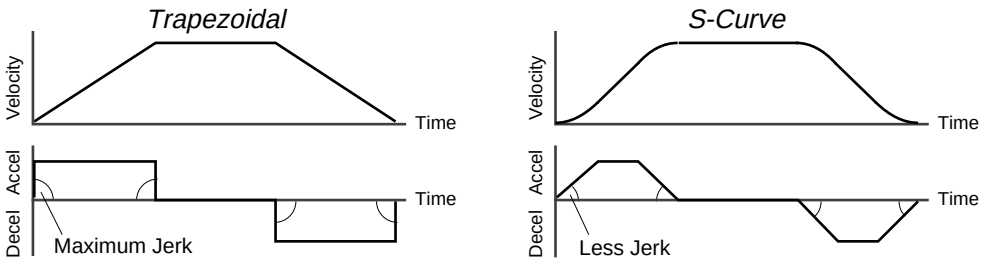
If the jog deceleration (JOGAD) command has not been entered, the jog acceleration (JOGA) command will also set the jog deceleration rate. Once the jog deceleration (JOGAD) command has been entered, the jog acceleration (JOGA) command no longer affects jog deceleration.

Example: Refer to the jog mode enable (JOG) command example.

JOGAA — Jogging Average Acceleration

		Product	Rev
Type	Motion (S-Curve)		
Syntax	<!><@><a>JOGAA<r>,<r>,<r>,<r>	AT6n00	n/a
Units	r = units/sec/sec	AT6n50	1.0
Range	0.00025 - 24999999 (depending on the scaling factor)	610n	n/a
Default	10.00 (trapezoidal profiling is default, where JOGAA tracks JOGA)	615n	1.0
Response	JOGAA: *JOGAA10.0000,10.0000,10.0000,10.0000 1JOGAA: *1JOGAA10.0000	620n	n/a
		625n	1.0
See Also	A, ADA, JOG, JOGA, JOGAD, JOGADA, SCALE, SCLA	6270	1.0

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



The values for the maximum jogging accel (JOGA) and average jogging accel (JOGAA) commands determine the characteristics of the S-curve. To smooth the acceleration ramp, you must enter a JOGAA command value that satisfies this equation: 1/2 JOGA ≤ JOGAA < JOGA. The following conditions are possible:

Acceleration Setting	Profiling Condition
JOGAA > 1/2 JOGA, but JOGAA < JOGA	S-curve profile with a variable period of constant acceleration
JOGAA = 1/2 JOGA	*Pure S-curve* (no period of constant acceleration—smoothest motion)
JOGAA = JOGA	Trapezoidal profile (but can be changed to an S-curve by specifying a new JOGAA value < JOGA)
JOGAA < 1/2 JOGA; or JOGAA > JOGA	When you issue the JOG command, the move will not be executed and an error message, *INVALID CONDITIONS FOR S_CURVE ACCELERATION–FIELD *n*, will be displayed.
JOGAA = zero	S-curve profiling is disabled. Trapezoidal profiling is enabled. JOGAA tracks JOGA, & JOGADA tracks JOGAD. (*Track* means the command's value will match the other command's value and will continue to match whatever the other command's value is set to.)
No JOGAA value ever entered	Profile will default to trapezoidal. JOGAA tracks JOGA.

While programming S-curves, if you never change the maximum or average jogging deceleration (JOGAD or JOGADA) commands, JOGADA will track JOGAA. However, once you change JOGAD, JOGADA will no longer track changes in JOGAA.

NOTE

Once you enter a JOGAA value that is ≠ zero or ≠ JOGA, S-curve profiling is enabled **only for jogging moves** (e.g., not for contouring, which requires the PADA and/or PAA commands). All subsequent jogging moves for that axis must comply with this equation: 1/2 JOGA ≤ JOGAA < JOGA.

Increasing the AA value above the pure S-curve level (JOGAA > 1/2 JOGA), the time required to reach the target velocity and the target distance is decreased. However, increasing JOGAA also increases jerk.

The calculation for determining S-curve average accel and decel move times is as follows:

(A_{avg} = average accel or decel value):

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

NOTE: Scaling (SCLA) affects JOGAA the same as it does for JOGA.

*** For a more in-depth discussion on S-curve profiling, refer to the Custom Profiling chapter in the *6000 Series Programmer's Guide*.

Example:

```
JOGA10,10,10,10    ; Sets the maximum jogging acceleration of all axes
JOGAA5,5,7.5,10    ; Sets the average jogging acceleration of all axes
```

JOGAD Jog Deceleration

		Product	Rev
Type	Jog		
Syntax	<!><@><a>JOGAD<r>,<r>,<r>,<r>	AT6n00	1.0
Units	r = units/sec/sec	AT6n50	1.0
Range	0.00025 - 24,999,999 (depending on the scaling factor)	610n	4.0
Default	10.0000 (JOGAD tracks JOGA)	615n	1.0
Response	JOGAD: *JOGAD10.0000,10.0000,10.0000,10.0000 1JOGAD: *1JOGAD10.0000	620n	1.0
		625n	1.0
See Also	DJOG, JOG, JOGA, JOGAA, JOGADA, JOGVH, JOGVL, INFNC, SCALE, SCLA	6270	1.0

The Jog Deceleration (JOGAD) command specifies the deceleration to be used when a jog input is released.

Steppers: The entered value is always in reference to motor steps, not encoder steps, regardless of the ENC command setting. If scaling is not enabled (SCALEØ), the deceleration value is entered in motor revs/sec/sec; this value is internally multiplied by the drive resolution (DRES) value to obtain an deceleration value in motor steps/sec/sec for the motion trajectory calculations. If scaling is enabled (SCALE1), the entered deceleration value is internally multiplied by the deceleration scaling factor (SCLA) to convert user units/sec/sec to motor steps/sec/sec.

Servos: If scaling is not enabled (SCALEØ), the deceleration value is entered in encoder or resolver revs/sec/sec, LDT inches/sec/sec, or ANI volts/sec/sec; encoder or resolver and LDT values are internally multiplied by the encoder resolution (ERES) value or the LDT resolution (LDTRES) value to obtain an acceleration value in steps/sec/sec for the motion trajectory calculations. If scaling is enabled (SCALE1), the entered deceleration value is internally multiplied by the deceleration scaling factor (SCLA) to convert user units/sec/sec to encoder, resolver, LDT, or ANI steps/sec/sec.

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

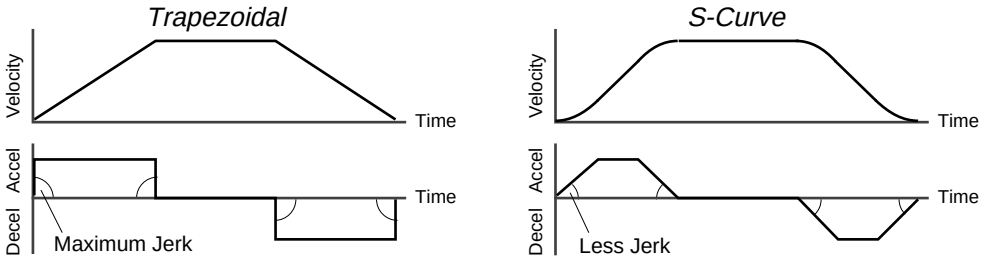
If the jog deceleration (JOGAD) command has not been entered, the jog acceleration (JOGA) command will also set the jog deceleration rate. Once the jog deceleration (JOGAD) command has been entered, the jog acceleration (JOGA) command no longer affects jog deceleration. If JOGAD is set to zero (JOGADØ), then the jog deceleration will once again track whatever the JOGA command is set to.

Example: Refer to the jog mode enable (JOG) command example.

JOGADA Jogging Average Deceleration

		Product	Rev
Type	Motion (S-Curve)		
Syntax	<!><@><a>JOGADA<r>,<r>,<r>,<r>	AT6n00	n/a
Units	r = units/sec/sec	AT6n50	1.0
Range	0.00025 - 24999999 (depending on the scaling factor)	610n	n/a
Default	10.00 (JOGADA tracks JOGAA)	615n	1.0
Response	JOGADA: *JOGADA10.0000,10.0000,10.0000,10.0000 1JOGADA: *1JOGADA10.0000	620n	n/a
		625n	1.0
See Also	A, AD, JOG, JOGA, JOGAA, JOGAD, SCALE, SCLA	6270	1.0

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



The values for the maximum jogging decel (JOGAD) and average jogging decel (JOGADA) commands determine the characteristics of the S-curve. To smooth the deceleration ramp, you must enter a JOGADA command value that satisfies this equation: 1/2 JOGAD ≤ JOGADA < JOGAD. The following conditions are possible:

Deceleration Setting	Profiling Condition
JOGADA > 1/2 JOGAD, but JOGADA < JOGAD	S-curve profile with a variable period of constant deceleration
JOGADA = 1/2 JOGAD	*Pure S-curve* (no period of constant deceleration—smoothest motion)
JOGADA = JOGAD	Trapezoidal profile (but can be changed to S-curve by specifying a new JOGADA value < JOGAD)
JOGADA < 1/2 JOGAD; or JOGADA > JOGAD	When you issue the JOG command, the move will not be executed and an error message, *INVALID CONDITIONS FOR S_CURVE ACCELERATION–FIELD *n*, will be displayed.
JOGADA = zero	Upon entering the JOGADAØ command, an error message, *INVALID DATA–FIELD *n*, will be displayed.
S-curve profiling with JOGAA, and no JOGADA or JOGAD ever entered	JOGADA will always match the JOGAA command value (identical S-curve accel and decel profiles). When you change JOGAD, JOGADA will no longer match changes in JOGAA.

NOTE

Once you enter a JOGADA value that is ≠ zero or ≠ JOGAD, S-curve profiling is enabled **only for jogging move decelerations** (e.g., not for contouring decelerations, which require the PADA command). All subsequent jogging moves for that axis must comply with this equation: 1/2 JOGAD ≤ JOGADA < JOGAD.

Increasing the JOGADA value above the pure S-curve level (JOGADA > 1/2 JOGAD), the time required to reach the target velocity and the target distance is decreased. However, increasing JOGADA also increases jerk.

The calculation for determining S-curve average accel and decel move times is as follows:

(A_{avg} = average accel or decel value):

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

NOTE: Scaling (SCLA) affects JOGADA the same as it does for JOGAD.

*** For a more in-depth discussion on S-curve profiling, refer to the Custom Profiling chapter in the *6000 Series Programmer's Guide*.

Example:

```
JOGAD10,10,10,10    ; Sets the maximum jog deceleration of all four axes
JOGADA5,5,7.5,10    ; Sets the average jog deceleration of all four axes
```

JOGVH Jog Velocity High

		Product	Rev
Type	Jog	AT6n00	1.0
Syntax	<!><@><a>JOGVH<r>,<r>,<r>,<r>	AT6n50	1.0
Units	r = units/sec (scalable with SCLV)	610n	4.0
Range	0.00000 - 1,600,000 (depending on the scaling factor & PULSE)	615n	1.0
Default	10.0000 (2.0000 for 6270 only)	620n	1.0
Response	JOGVH: *JOGVH10.0000,10.0000,10.0000,10.0000 1JOGVH: *1JOGVH10.0000	625n	1.0
See Also	DJOG, JOG, JOGA, JOGAA, JOGAD, JOGADA, JOGVL, INFNC, PULSE, SCALE, SCLV	6270	1.0

The Jog Velocity High (JOGVH) command specifies the velocity to be used upon receiving a jog input with the jog velocity select input active (*ON*).

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

Steppers: The entered value is always in reference to motor steps, not encoder steps, regardless of the ENC command setting. If scaling is not enabled (SCALEØ), the velocity value is entered in motor revs/sec; this value is internally multiplied by the drive resolution (DRES) value to obtain a velocity value in motor steps/sec for the motor trajectory calculations. If scaling is enabled (SCALE1), the entered velocity value is internally multiplied by the velocity scaling factor (SCLV) to convert user units/sec to motor steps/sec.

Servos: If scaling is not enabled (SCALEØ), the velocity value is entered in encoder revs/sec, LDT inches/sec, or ANI volts/sec; encoder and LDT values are internally multiplied by the encoder resolution (ERES) value or the LDT resolution (LDTRES) value to obtain a velocity value in steps/sec for the motion trajectory calculations. If scaling is enabled (SCALE1), the entered velocity value is internally multiplied by the velocity scaling factor (SCLV) to convert user units/sec to encoder, LDT, or ANI steps/sec.

Example: Refer to the jog mode enable (JOG) command example.

JOGVL Jog Velocity Low

		Product	Rev
Type	Jog	AT6n00	1.0
Syntax	<!><@><a>JOGVL<r>,<r>,<r>,<r>	AT6n50	1.0
Units	r = units/sec (scalable with SCLV)	610n	4.0
Range	0.00000 - 1,600,000 (depending on the scaling factor & PULSE)	615n	1.0
Default	0.5000	620n	1.0
Response	JOGVL: *JOGVL.50000,.50000,.50000,.50000 1JOGVL: *1JOGVL5.0000	625n	1.0
See Also	DJOG, JOG, JOGA, JOGAA, JOGAD, JOGADA, JOGVH, INFNC, PULSE, SCALE, SCLV	6270	1.0

The Jog Velocity Low (JOGVL) command specifies the velocity to be used upon receiving a jog input with the jog velocity select input low, or *OFF*. The velocity remains set until you change it with a subsequent jog velocity low command. Velocities outside the valid range are flagged as an error, with a message *INVALID DATA-FIELD x, where x is the field number. When an invalid velocity is entered the previous velocity value is retained.

Steppers: The entered value is always in reference to motor steps, not encoder steps, regardless of the ENC command setting. If scaling is not enabled (SCALEØ), the velocity value is entered in motor revs/sec; this value is internally multiplied by the drive resolution (DRES) value to obtain a velocity value in motor steps/sec for the motor trajectory calculations. If scaling is enabled (SCALE1), the entered velocity value is internally multiplied by the velocity scaling factor (SCLV) to convert user units/sec to motor steps/sec.

Servos: If scaling is not enabled (SCALEØ), the velocity value is entered in encoder revs/sec, LDT inches/sec, or ANI volts/sec; encoder and LDT values are internally multiplied by the encoder resolution (ERES) value or the LDT resolution (LDTRES) value to obtain a velocity value in steps/sec for the motion trajectory calculations. If scaling is enabled (SCALE1), the entered velocity value is internally multiplied by the velocity scaling factor (SCLV) to convert user units/sec to encoder, LDT, or ANI steps/sec.

Example: Refer to the jog mode enable (JOG) command example.

JOY — Joystick Mode Enable

Type	Joystick
Syntax	<!><@><a>JOY<b><b><b><b>
Units	n/a
Range	b = 0 (disable), 1 (enable), or X (don't change)
Default	0
Response	JOY: *JOY0000 1JOY: *1JOY0
See Also	ANVO, ANVOEN, [AS], COMEXC, [INO], JOYA, JOYAA, JOYAD, JOYADA, JOYAXH, JOYAXL, JOYCDB, JOYCTR, JOYEDB, JOYVH, JOYVL, JOYZ, TAS, TINO

Product	Rev
AT6n00	1.0
OEM-AT6n00	n/a
AT6n50	1.0
610n	n/a
615n	n/a
620n	1.0
625n	1.0
6270	1.0

This command enables joystick mode on the appropriate axes. Once joystick mode has been enabled, the analog inputs can be used to produce motion on a specific axis. Motion will be directly proportional to the voltage on the analog inputs, which is linearly related to the joystick positioning. All command processing will stop (assuming COMEXCØ) until the joystick release input becomes active, or an immediate joystick disable (!JOYØØØØ) command is issued. Enabling joystick mode for a specific axis places that axis in a moving condition; no further motion commands (GO) can be executed for that axis while in joystick mode, unless the continuous command execution mode is enabled (COMEXC1).

There are several other inputs available on the joystick 25-pin "D" connector, in addition to the analog channels. The connections are shown below, along with a description of the function for each input.

Pin # on the 25-pin Joystick Connector	Function
1	Analog Channel 1
2	Analog Channel 2
3	Analog Channel 3
4	Analog Channel 4 *(AT6400 and AT6n50)*
8	Shield
14	Ground

Pin # on the 25-pin Joystick Connector	Function
15	Axes Select
16	Velocity Select
17	Joystick Release
18	Joystick Trigger
19	Joystick Auxiliary
23	+5VDC (out)

The axes select input determines the axes that the joystick will control. The axes that correspond to the input when it is active are determined by the JOYAXH command. The axes that correspond to the input when it is inactive are determined by the JOYAXL command.

The velocity select input determines the maximum velocity when the joystick is at full deflection. The velocity that corresponds to the input when it is active is determined by the JOYVH command. The velocity that corresponds to the input when it is inactive is determined by the JOYVL command.

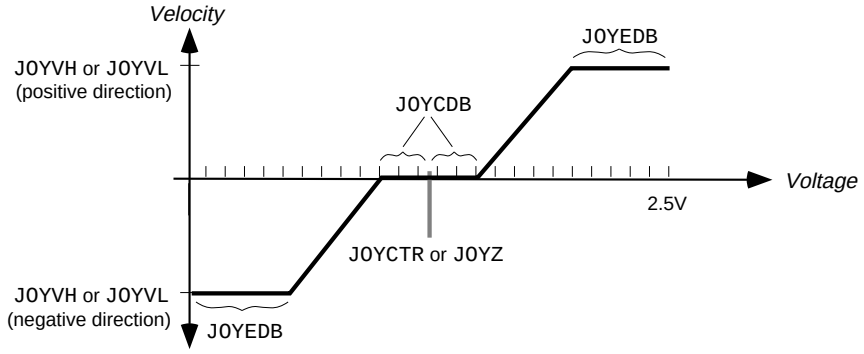
The joystick release input disables joystick mode on all axes (same as issuing !JOYØØØØ). The joystick release input requires a normally-closed switch that disables the joystick mode when the switch is opened.

The auxiliary input and the joystick trigger input can be used as additional inputs. Use the TINO and INO commands to read the level of these inputs.

The valid voltage range for the analog inputs is 0 - 2.5 VDC, as applied between an analog input and ground. For positive-direction motion, a 1.26 to 2.5 VDC signal is required. For negative-direction motion, a 0 to 1.24 VDC signal is required. For best results, a 5KΩ single-turn joystick potentiometer with a 1KΩ pull-up resistor is recommended; since most joysticks allow only 60° of travel (20% of the potentiometer's range), adjust the potentiometer so that full deflection of the joystick moves the potentiometer from 0KΩ to 1KΩ. The 6000 Series product's internal analog input circuit is provided in the product's *Installation Guide*.

Example (refer also to the illustration below):

```
JOYAXH1,2,0,0     ; When axis select input is high (active), analog channel 1
                  ; controls axis 1, analog channel 2 controls axis 2,
                  ; and analog channels 3 & 4 do not control any axis
JOYAXL0,0,1,2     ; When axis select input is low (inactive), analog channel 1
                  ; controls axis 3, analog channel 2 controls axis 4,
                  ; and analog channels 3 & 4 do not control any axis
@JOYA100          ; Set joystick acceleration to 100 units/sec/sec on all axes
@JOYAD000         ; Set joystick deceleration to 200 units/sec/sec on all axes
@JOYCDB0.25       ; Set center deadband to +/-0.25V for all analog channels
@JOYEDB0.5        ; Set end deadband to 0.5V for all analog channels
                  ; (limits usable voltage range to 0.5 - 2.0V)
@JOYCTR1.25,1.25  ; Set joystick center at 1.25 volts for all analog channels
@JOYVH10          ; Set velocity to 10 units/sec on all axes
                  ; (applies when the joystick velocity select input is high)
JOYVL1,1,2,2      ; Set velocity to 1 unit/sec on axes 1 and 2,
                  ; 2 unit/sec on axes 3 and 4
                  ; (applies when the joystick velocity select input is low)
JOY1111           ; Enable joystick mode on all axes.
                  ; Toggling the axis select input on the joystick connector
                  ; will cause analog inputs 1 and 2 to control axes 1 and 2
                  ; in one state, and axes 3 and 4 in the other.
```



JOYA — Joystick Acceleration

Type	Joystick
Syntax	<!><@><a>JOYA<r>,<r>,<r>,<r>
Units	r = units/sec/sec
Range	0.07500 - 24,999,999 (depending on the scaling factor)
Default	10.0000
Response	JOYA: *JOYA10.0000,10.0000,10.0000,10.0000 1JOYA: *1JOYA10.0000
See Also	JOY, JOYAA, JOYAD, JOYADA, JOYAXH, JOYAXL, JOYCDB, JOYCTR, JOYEDB, JOYVH, JOYVL, JOYZ, SCALE, SCLA

Product	Rev
AT6n00	1.0
OEM-AT6n00	n/a
AT6n50	1.0
610n	n/a
615n	n/a
620n	1.0
625n	1.0
6270	1.0

The Joystick Acceleration (JOYA) command specifies the acceleration to be used during joystick mode.

Steppers: The entered value is always in reference to motor steps, not encoder steps, regardless of the ENC command setting. If scaling is not enabled (SCALEØ), the acceleration value is entered in motor revs/sec/sec; this value is internally multiplied by the drive resolution (DRES) value to obtain an acceleration value in motor steps/sec/sec for the motion trajectory calculations. If scaling is enabled (SCALE1), the entered acceleration value is internally multiplied by the acceleration scaling factor (SCLA) to convert user units/sec/sec to motor steps/sec/sec.

Servos: If scaling is not enabled (SCALEØ), the acceleration value is entered in encoder or resolver revs/sec/sec, LDT inches/sec/sec, or ANI volts/sec/sec; encoder or resolver and LDT values are internally multiplied by the encoder resolution (ERES) value or the LDT resolution (LDTRES) value to obtain an acceleration value in steps/sec/sec for the motion trajectory calculations. If scaling is enabled (SCALE1), the entered acceleration value is internally multiplied by the acceleration scaling factor (SCLA) to convert user units/sec/sec to encoder, resolver, LDT, or ANI steps/sec/sec.

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

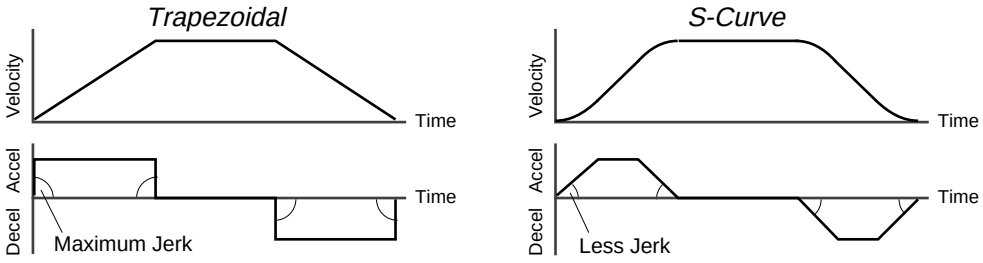
If the joystick deceleration (JOYAD) command has not been entered, the joystick acceleration (JOYA) command will also set the joystick deceleration rate. Once the joystick deceleration (JOYAD) command has been entered, the joystick acceleration (JOYA) command no longer affects joystick deceleration.

Example: Refer to the joystick mode enable (JOY) command example.

JOYAA Joystick Average Acceleration

		Product	Rev
Type	Motion (S-Curve)	AT6n00	n/a
Syntax	<!><@><a>JOYAA<r>,<r>,<r>,<r>	AT6n50	1.0
Units	r = units/sec/sec	610n	n/a
Range	0.00025 - 24999999 (depending on the scaling factor)	615n	n/a
Default	10.00 (trapezoidal profiling is default, where JOYAA tracks JOYA)	620n	n/a
Response	JOYAA: *JOYAA10.0000,10.0000,10.0000,10.0000 1JOYAA: *1JOYAA10.0000	625n	1.0
See Also	AA, AD, JOY, JOYA, JOYAD, JOYADA, SCALE, SCLA	6270	1.0

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



The values for the maximum joystick accel (JOYA) and average joystick accel (JOYAA) commands determine the characteristics of the S-curve. To smooth the acceleration ramp, you must enter a JOYAA command value that satisfies this equation: 1/2 JOYA ≤ JOYAA < JOYA. The following conditions are possible:

Acceleration Setting	Profiling Condition
JOYAA > 1/2 JOYA, but JOYAA < JOYA	S-curve profile with a variable period of constant acceleration
JOYAA = 1/2 JOYA	*Pure S-curve* (no period of constant acceleration—smoothest motion)
JOYAA = JOYA	Trapezoidal profile (but can be changed to an S-curve by specifying a new JOYAA value < JOYA)
JOYAA < 1/2 JOYA; or JOYAA > JOYA	When you issue the JOY command, the move will not be executed and an error message, *INVALID CONDITIONS FOR S_CURVE ACCELERATION–FIELD *n*, will be displayed.
JOYAA = zero	S-curve profiling is disabled. Trapezoidal profiling is enabled. JOYAA tracks JOYA, & JOYADA tracks JOYAD. (*Track* means the command's value will match the other command's value and will continue to match whatever the other command's value is set to.)
No JOYAA value ever entered	Profile will default to trapezoidal. JOYAA tracks JOYA.

While programming S-curves, if you never change the maximum or average joystick deceleration (JOYAD or JOYADA) commands, JOYADA will track JOYAA. However, once you change JOYAD, JOYADA will no longer track changes in JOYAA.

NOTE

Once you enter a JOYAA value that is ≠ zero or ≠ JOYA, S-curve profiling is enabled **only for joystick moves** (e.g., not for contouring, which requires the PADA and/or PAA commands). All subsequent joystick moves for that axis must comply with this equation: 1/2 JOYA ≤ JOYAA < JOYA.

Increasing the AA value above the pure S-curve level (JOYAA > 1/2 JOYA), the time required to reach the target velocity and the target distance is decreased. However, increasing JOYAA also increases jerk.

The calculation for determining S-curve average accel and decel move times is as follows:

(A_{avg} = average accel or decel value):

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

NOTE: Accelerating Scaling (SCLA) affects JOYAA the same as it does for JOYA.

*** For a more in-depth discussion on S-curve profiling, refer to the Custom Profiling chapter in the *6000 Series Programmer's Guide*.

Example:

```
JOYA10,10,10,10   ; Set the maximum joystick acceleration of all four axes
JOYAA5,5,7.5,10   ; Set the average joystick acceleration of all four axes
```

JOYAD Joystick Deceleration

Type	Joystick
Syntax	<!><@><a>JOYAD<r>,<r>,<r>,<r>
Units	r = units/sec/sec
Range	0.07500 - 24,999,999 (depending on the scaling factor)
Default	10.0000 (JOYAD tracks JOYA)
Response	JOYAD: *JOYAD10.0000,10.0000,10.0000,10.0000 1JOYAD: *1JOYAD10.0000
See Also	JOY, JOYA, JOYAA, JOYADA, JOYAXH, JOYAXL, JOYCDB, JOYCTR, JOYEDB, JOYVH, JOYVL, JOYZ, SCALE, SCLA

Product	Rev
AT6n00	1.0
OEM-AT6n00	n/a
AT6n50	1.0
610n	n/a
615n	n/a
620n	1.0
625n	1.0
6270	1.0

The Joystick Deceleration (JOYAD) command specifies the deceleration to be used during the joystick mode.

Steppers: The entered value is always in reference to motor steps, not encoder steps, regardless of the ENC command setting. If scaling is not enabled (SCALEØ), the deceleration value is entered in motor revs/sec/sec; this value is internally multiplied by the drive resolution (DRES) value to obtain an deceleration value in motor steps/sec/sec for the motion trajectory calculations. If scaling is enabled (SCALE1), the entered deceleration value is internally multiplied by the deceleration scaling factor (SCLA) to convert user units/sec/sec to motor steps/sec/sec.

Servos: If scaling is not enabled (SCALEØ), the deceleration value is entered in encoder or resolver revs/sec/sec, LDT inches/sec/sec, or ANI volts/sec/sec; encoder or resolver and LDT values are internally multiplied by the encoder resolution (ERES) value or the LDT resolution (LDTRES) value to obtain an acceleration value in steps/sec/sec for the motion trajectory calculations. If scaling is enabled (SCALE1), the entered deceleration value is internally multiplied by the deceleration scaling factor (SCLA) to convert user units/sec/sec to encoder, resolver, LDT, or ANI steps/sec/sec.

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

If the joystick deceleration (JOYAD) command has not been entered, the joystick acceleration (JOYA) command will also set the joystick deceleration rate. Once the joystick deceleration (JOYAD) command has been entered, the joystick acceleration (JOYA) command no longer affects joystick deceleration. If JOYAD is set to zero (JOYADØ), then the joystick deceleration will once again track whatever the JOYA command is set to.

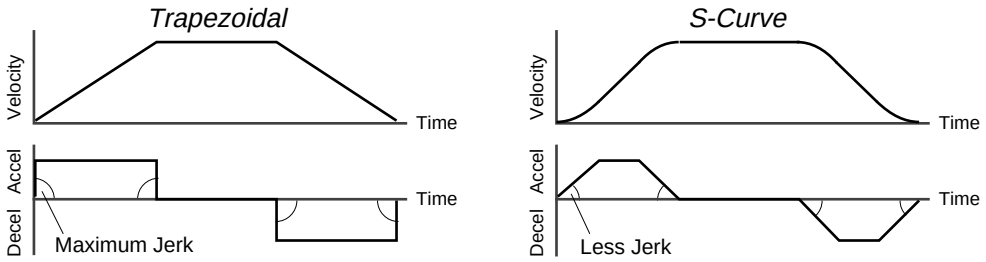
Example: Refer to the joystick mode enable (JOY) command example.

JOYADA Joystick Average Deceleration

Type	Motion (S-Curve)
Syntax	<!><@><a>JOYADA<r>,<r>
Units	r = units/sec/sec
Range	0.00025 - 24999999 (depending on the scaling factor)
Default	10.00 (JOYADA tracks JOYAA)
Response	JOYADA: *JOYADA10.0000,10.0000 1JOYADA: *1JOYADA10.0000
See Also	A, AD, JOY, JOYA, JOYAA, JOYAD, SCALE, SCLA

Product	Rev
AT6n00	n/a
AT6n50	1.0
610n	n/a
615n	n/a
620n	n/a
625n	1.0
6270	1.0

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



The values for the maximum joystick decel (JOYAD) and average joystick decel (JOYADA) commands determine the characteristics of the S-curve. To smooth the deceleration ramp, you must enter a JOYADA command value that satisfies this equation: 1/2 JOYAD ≤ JOYADA < JOYAD. The following conditions are possible:

Deceleration Setting	Profiling Condition
JOYADA > 1/2 JOYAD, but JOYADA < JOYAD	S-curve profile with a variable period of constant deceleration
JOYADA = 1/2 JOYAD	*Pure S-curve* (no period of constant deceleration—smoothest motion)
JOYADA = JOYAD	Trapezoidal profile (but can be changed to S-curve by specifying a new JOYADA value < JOYAD)
JOYADA < 1/2 JOYAD; or JOYADA > JOYAD	When you issue the JOY command, the move will not be executed and an error message, *INVALID CONDITIONS FOR S_CURVE ACCELERATION–FIELD *n*, will be displayed.
JOYADA = zero	Upon entering the JOYADAØ command, an error message, *INVALID DATA–FIELD *n*, will be displayed.
S-curve profiling with JOYAA, and no JOYADA or JOYAD ever entered	JOYADA will always match the JOYAA command value (identical S-curve accel and decel profiles). When you change JOYAD, JOYADA will no longer match changes in JOYAA.

NOTE

Once you enter a JOYADA value that is ≠ zero or ≠ JOYAD, S-curve profiling is enabled **only for joystick move decelerations** (e.g., not for contouring decelerations, which require the PADA command). All subsequent joystick moves for that axis must comply with this equation: 1/2 JOYAD ≤ JOYADA < JOYAD.

Increasing the JOYADA value above the pure S-curve level (JOYADA > 1/2 JOYAD), the time required to reach the target velocity and the target distance is decreased. However, increasing JOYADA also increases jerk.

The calculation for determining S-curve average accel and decel move times is as follows:

(A_{avg} = average accel or decel value):

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

Acceleration Scaling (SCLA) affects JOYADA the same as it does for JOYAD.

*** For a more in-depth discussion on S-curve profiling, refer to the Custom Profiling chapter in the *6000 Series Programmer's Guide*.

Example:

```
JOYAD10,10,10,10    ; Sets the maximum joystick deceleration of all four axes
JOYADA5,5,7.5,10    ; Sets the average joystick deceleration of all four axes
```

JOYAXH Joystick Analog Channel High

Type	Joystick
Syntax	<!><@><a>JOYAXH<i>,<i>,<i>,<i>
Units	i = analog channel #
Range	0 - 4 (AT6n00 & AT6n50); 0 - 3 (620n, 625n & 6270)
Default	1,2,3,4 (AT6n00 & AT6450); 1,2 (AT6250, 620n, 625n & 6270)
Response	JOYAXH: *JOYAXH1,2,3,4 1JOYAXH: *1JOYAXH1
See Also	JOY, JOYA, JOYAA, JOYAD, JOYADA, JOYAXL, JOYCDB, JOYCTR, JOYEDB, JOYVH, JOYVL, JOYZ

Product	Rev
AT6n00	1.0
OEM-AT6n00	n/a
AT6n50	1.0
610n	n/a
615n	n/a
620n	1.0
625n	1.0
6270	1.0

The Joystick Analog Channel High (JOYAXH) command specifies the analog channel that will control each axis during joystick mode, while the joystick axes select input is high (sinking current) and the corresponding axis is in joystick mode. A single analog input channel can control more than one axis (e.g., JOYAXH1,1,Ø,Ø). The value entered in the command field is an analog channel number. If zero is specified, no analog channel will control the corresponding axis when the axes select input is high.

Example: Refer to the joystick mode enable (JOY) command example.

JOYAXL Joystick Analog Channel Low

Type	Joystick
Syntax	<!><@><a>JOYAXL<i>,<i>,<i>,<i>
Units	i = analog channel #
Range	0 - 4 (AT6n00 & AT6n50); 0 - 3 (620n, 625n & 6270)
Default	1,2,3,4 (AT6n00 & AT6450); 1,2 (AT6250, 620n, 625n & 6270)
Response	JOYAXL: *JOYAXL1,2,3,4 1JOYAXL: *1JOYAXL1
See Also	JOY, JOYA, JOYAA, JOYAD, JOYADA, JOYAXH, JOYCDB, JOYCTR, JOYEDB, JOYVH, JOYVL, JOYZ

Product	Rev
AT6n00	1.0
OEM-AT6n00	n/a
AT6n50	1.0
610n	n/a
615n	n/a
620n	1.0
625n	1.0
6270	1.0

The Joystick Analog Channel Low (JOYAXL) command specifies the analog channel that will control each axis during joystick mode, while the joystick axes select input is low (not sinking current) and the corresponding axis is in joystick mode. A single analog input channel can control more than one axis (e.g., JOYAXL1,1,Ø,Ø). If zero is specified, no analog channel will control the corresponding axis when the axes select input is low.

Example: Refer to the joystick mode enable (JOY) command example.

JOYCDB Joystick Center Deadband

Type	Joystick
Syntax	<!><@><a>JOYCDB<r>,<r>,<r>,<r>
Units	r = volts
Range	0.00 - 1.24
Default	0.1000
Response	JOYCDB: *JOYCDB0.1000,0.1000,0.10000,0.1000 1JOYCDB: *1JOYCDB0.1000
See Also	JOY, JOYA, JOYAA, JOYAD, JOYADA, JOYAXH, JOYAXL, JOYCTR, JOYEDB, JOYVH, JOYVL, JOYZ

Product	Rev
AT6n00	1.0
OEM-AT6n00	n/a
AT6n50	1.0
610n	n/a
615n	n/a
620n	1.0
625n	1.0
6270	1.0

The Joystick Center Deadband (JOYCDB) command sets the range of voltage about the joystick center (JOYCTR) which will command no motion. Each analog channel can have a JOYCDB value between 0.00V and 1.24V, with the resolution being 2 decimal places. ***NOTE*: The data fields represent the analog channels, not the axes.**

Example: Refer to the joystick mode enable (JOY) command example.

JOYCTR Joystick Center

		Product	Rev
Type	Joystick	AT6n00	1.0
Syntax	<!><@><a>JOYCDB<r>,<r>,<r>,<r>	OEM-AT6n00	n/a
Units	r = volts	AT6n50	1.0
Range	0.15 - 2.40	610n	n/a
Default	1.2500	615n	n/a
Response	JOYCTR: *JOYCTR1.2500,1.2500,1.2500,1.2500 1JOYCTR: *1JOYCTR1.25	620n	1.0
See Also	JOY, JOYA, JOYAA, JOYAD, JOYADA, JOYAXH, JOYAXL, JOYCDB, JOYEDB, JOYVH, JOYVL, JOYZ	625n	1.0
		6270	1.0

The Joystick Center (JOYCTR) command defines the voltage level for the analog input that commands no motion for the analog input. The fields <r> after the JOYCTR command correspond to analog inputs. The resolution of the JOYCTR command is 2 decimal places. ***NOTE*: The data fields represent the analog channels, not the axes.**

Example: Refer to the joystick mode enable (JOY) command example.

JOYEDB Joystick End Deadband

		Product	Rev
Type	Joystick	AT6n00	1.0
Syntax	<!><@><a>JOYEDB<r>,<r>,<r>,<r>	OEM-AT6n00	n/a
Units	r = volts	AT6n50	1.0
Range	0.00 - 1.24	610n	n/a
Default	0.1000	615n	n/a
Response	JOYEDB: *JOYCEB0.1000,0.1000,0.10000,0.1000 1JOYEDB: *1JOYEDB0.1000	620n	1.0
See Also	JOY, JOYA, JOYAA, JOYAD, JOYADA, JOYAXH, JOYAXL, JOYCDB, JOYCTR, JOYVH, JOYVL, JOYZ	625n	1.0
		6270	1.0

The Joystick End Deadband (JOYEDB) command defines the voltage that is used to determine the full voltage range of the analog inputs. By specifying an end deadband, you are effectively decreasing the voltage range over which the analog input will function.

This command is useful if your joystick does not reach either limit of the voltage range (0.00V to 2.50V). This statement reduces the range to fit the voltage range for that joystick. The resolution of the JOYEDB command is 2 decimal places. ***NOTE*: The data fields represent the analog channels, not the axes.**

Example: Refer to the joystick mode enable (JOY) command example.

JOYVH Joystick Velocity High

		Product	Rev
Type	Joystick	AT6n00	1.0
Syntax	<!><@><a>JOYVH<r>,<r>,<r>,<r>	OEM-AT6n00	n/a
Units	r = units/sec	AT6n50	1.0
Range	0.00020 - 1,600,000 (depending on the scaling factor & PULSE)	610n	n/a
Default	0.5000	615n	n/a
Response	JOYVH: *JOYVH0.5000,0.5000,0.5000,0.5000 1JOYVH: *1JOYVH0.5000	620n	1.0
See Also	JOY, JOYA, JOYAA, JOYAD, JOYADA, JOYAXH, JOYAXL, JOYCDB, JOYCTR, JOYEDB, JOYVL, JOYZ, PULSE. SCALE, SCLV	625n	1.0
		6270	1.0

The Joystick Velocity High (JOYVH) command specifies the maximum velocity that can be obtained at full deflection during joystick mode, with the joystick velocity select input high (sinking current). ***NOTE***: The data fields (<r>,<r>,<r>,<r>) represent the axes, **not the analog channels**.

The joystick velocity must be entered prior to entering joystick mode (JOY). The joystick velocity high remains set until you change it with a subsequent JOYVH command. Velocities outside the valid range are flagged as an error, with a message *INVALID DATA-FIELD x, where x is the field number. When an invalid velocity is entered the previous velocity value is retained.

Steppers: The entered value is always in reference to motor steps, not encoder steps, regardless of the ENC command setting. If scaling is not enabled (SCALEØ), the velocity value is entered in motor revs/sec; this value is internally multiplied by the drive resolution (DRES) value to obtain a velocity value in motor steps/sec for the motor trajectory calculations. If scaling is enabled (SCALE1), the entered velocity value is internally multiplied by the velocity scaling factor (SCLV) to convert user units/sec to motor steps/sec.

Servos: If scaling is not enabled (SCALEØ), the velocity value is entered in encoder revs/sec, LDT inches/sec, or ANI volts/sec; encoder and LDT values are internally multiplied by the encoder resolution (ERES) value or the LDT resolution (LDTRES) value to obtain a velocity value in steps/sec for the motion trajectory calculations. If scaling is enabled (SCALE1), the entered velocity value is internally multiplied by the velocity scaling factor (SCLV) to convert user units/sec to encoder, LDT, or ANI steps/sec.

Example: Refer to the joystick mode enable (JOY) command example.

JOYVL Joystick Velocity Low

Type	Joystick
Syntax	<!><@><a>JOYVL<r>,<r>,<r>,<r>
Units	r = units/sec
Range	0.00020 - 1,600,000 (depending on the scaling factor & PULSE
Default	0.2000
Response	JOYVL: *JOYVL0.2000,0.2000,0.2000,0.2000 1JOYVL: *1JOYVL0.2000
See Also	JOY, JOYA, JOYAA, JOYAD, JOYADA, JOYAXH, JOYAXL, JOYCDB, JOYCTR, JOYEDB, JOYVH, JOYZ, PULSE, SCALE, SCLV

Product	Rev
AT6n00	1.0
OEM-AT6n00	n/a
AT6n50	1.0
610n	n/a
615n	n/a
620n	1.0
625n	1.0
6270	1.0

The Joystick Velocity Low (JOYVL) command specifies the maximum velocity that can be obtained at full deflection during joystick mode, with the joystick velocity select input low (not sinking current). ***NOTE***: The data fields (<r>,<r>,<r>,<r>) represent the axes, **not the analog channels**.

The joystick velocity must be entered prior to entering joystick mode (JOY). The joystick velocity low remains set until you change it with a subsequent JOYVL command. Velocities outside the valid range are flagged as an error, with a message *INVALID DATA-FIELD x, where x is the field number. When an invalid velocity is entered the previous velocity value is retained.

Steppers: The entered value is always in reference to motor steps, not encoder steps, regardless of the ENC command setting. If scaling is not enabled (SCALEØ), the velocity value is entered in motor revs/sec; this value is internally multiplied by the drive resolution (DRES) value to obtain a velocity value in motor steps/sec for the motor trajectory calculations. If scaling is enabled (SCALE1), the entered velocity value is internally multiplied by the velocity scaling factor (SCLV) to convert user units/sec to motor steps/sec.

Servos: If scaling is not enabled (SCALEØ), the velocity value is entered in encoder revs/sec, LDT inches/sec, or ANI volts/sec; encoder and LDT values are internally multiplied by the encoder resolution (ERES) value or the LDT resolution (LDTRES) value to obtain a velocity value in steps/sec for the motion trajectory calculations. If scaling is enabled (SCALE1), the entered velocity value is internally multiplied by the velocity scaling factor (SCLV) to convert user units/sec to encoder, LDT, or ANI steps/sec.

Example: Refer to the joystick mode enable (JOY) command example.

JOYZ — Joystick Zero

Type	Joystick
Syntax	<!><@>JOYZ<b><b><b><b>
Units	n/a
Range	b = 0 (don't zero), 1 (zero), or X (don't change)
Default	n/a
Response	n/a
See Also	JOY, JOYA, JOYAA, JOYAD, JOYADA, JOYAXH, JOYAXL, JOYCDB, JOYCTR, JOYEDB, JOYVH, JOYVL

Product	Rev
AT6n00	1.0
OEM-AT6n00	n/a
AT6n50	1.0
610n	n/a
615n	n/a
620n	1.0
625n	1.0
6270	1.0

The Joystick Zero (JOYZ) command defines the current analog input voltage on the enabled axes as center. If the command JOYZ11ØØ was issued, the current voltage on analog channels 1 and 2 would be read in and considered the joystick center, where no motion will occur. This command will automatically determine center voltages, thus eliminating the need for the JOYCTR command.

NOTE: The data fields (bbbb) represent the corresponding analog channels, **not the axes**.

Example:

```
JOYAXH4,2,2,3     ; Set the analog input channel each axis is to use when the
                  ; joystick axis select input is high (axis 1 uses analog
                  ; channel 4, axes 2 & 3 use analog channel 2, and axis 4 uses
                  ; analog channel 3)
@JOYAXL1          ; All axes use analog input channel 1 when the joystick
                  ; axis select input is low
JOYA100,100       ; Set joystick acceleration to 100 units/sec/sec on axes 1 & 2
JOYAD200,200      ; Set joystick deceleration to 200 units/sec/sec on axes 1 & 2
JOYCDBØ.25,0.25   ; Set center deadband to ± 0.25 volts on axes 1 & 2
JOYEDBØ.5,0.5     ; Set end deadband to 0.5 volts on axes 1 & 2 (allows for a
                  ; voltage of 0.5 to 2.0 volts for the analog input)
JOYZ1100          ; Automatically set the center voltage for analog chnls 1 & 2
JOYVH10,10        ; Set velocity to 10 on axes 1 & 2 when the joystick
                  ; velocity select input is high
JOYVL1,1          ; Set velocity to 1 on axes 1 & 2 when the joystick
                  ; velocity select input is low
JOY1100           ; Enable joystick mode on axes 1 & 2
```

JUMP — Jump to a Program or Label (and do not return)

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

Product	Rev
AT6n00	2.2
AT6n50	1.0
610n	4.0
615n	1.0
620n	2.1
625n	1.1
6270	1.0

The JUMP command branches to the corresponding program name or label when executed. A program or label name consists of 6 or fewer alpha-numeric characters.

All nested IFs, WHILEs, and REPEATs, loops, and subroutines are cleared; thus, the program or label that the JUMP initiates will **not** return control to the line after the JUMP, when the program completes operation. Instead, the program will end.

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

Example

```
; ************************************************************************
; * In this example, the program place is executed and calls the pick   *
; * subroutine. The pick subroutine then initiates motion on axes 1 & 2 *
; * (GO1100) and jumps to the program called load to initiate motion on *
; * axis 3 (GO001). Then, because the JUMP command cleared the pick     *
; * subroutine, program execution is terminated instead of returning to *
; * the place program.                                                  *
; ************************************************************************
```

```
DEF pick        ; Begin definition of subroutine named pick
GO1100          ; Initiate motion on axes 1 and 2
JUMP load       ; Jump to the program named load
END             ; End subroutine definition
DEF load        ; Begin definition of program named load
GO001           ; Initiate motion on axis 3
END             ; End program definition
DEF place       ; Begin definition of subroutine named place
GOSUB pick      ; Gosub to subroutine named pick
GO1000          ; Initiate motion on axis 1
END             ; End subroutine definition
RUN place       ; Execute program named place
```

K — Kill Motion

Type	Motion
Syntax	<!><@>K<b><b><b><b>
Units	n/a
Range	b = 0 (don't kill), 1 (kill), or X (don't change)
Default	n/a
Response	!K No response, instead motion is killed on all axes
See Also	COMEXK, DRFLVL, FOLK, GO, <CTRL>K, KDRIVE, LHAD, LHADA, S, TAS

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Kill Motion (K) command instructs the motor to instantaneously stop motion on the specified axes. If the Kill (K) command is used without any arguments (K or !K), motion will be stopped on all axes, and program execution will be terminated. When the Kill (K) command is used with ones in the command fields (e.g., KØ11Ø), motion will be stopped on the axes specified with ones (1), and program execution will continue with the next command. The Kill command will be used most frequently with the immediate command delimiter in front of the command (!K). By using the immediate Kill (!K) command, motion will be stopped at the time the command is received.

STEPPER SYSTEMS

This command should be used with caution. Since motion is stopped instantaneously, without a controlled deceleration ramp, high inertial loads may cause a drive to fault. A drive fault condition will allow the load to *free wheel*, possibly damaging equipment. Compumotor recommends using a brake on your motor drive system to brake the load in the event of a drive fault.

Servos: Motion is stopped at the rate set with the LHADA and LHAD commands. If you want the drive to be disabled upon executing a K or !K command, enable the *Disable Dive on Kill* mode with the KDRIVE command. (**CAUTION**: In the KDRIVE mode, a K or !K command immediately shuts down the drive, allowing the load to *free wheel* to a stop.)

Example:

```
A2,2,25000,25000  ; Set acceleration to 2, 2, 25000, and 25000 units/sec/sec
                  ; for axes 1, 2, 3 and 4
AD2,2,25000,25000 ; Set deceleration to 2, 2, 25000, and 25000 units/sec/sec
                  ; for axes 1, 2, 3 and 4
V1,1,1,2          ; Set velocity to 1, 1, 1, and 2 units/sec for axes 1, 2, 3
                  ; and 4, respectively
@D10              ; Set distance on all axes to 10 units
@GO1              ; Initiate motion on all axes -- motion begins.
                  ; After a short period the Kill command is sent.
!K                ; Kill motion on all axes (steppers stop instantaneously,
                  ; servos stop at the LHADA/LHAD decel)
```

<CTRL>K Kill Motion

		Product	Rev
Type	Motion	AT6n00	1.0
Syntax	<CTRL>K	AT6n50	1.0
Units	n/a	610n	4.0
Range	n/a	615n	1.0
Default	n/a	620n	1.0
Response	<CTRL>K: No response, instead motion is killed on all axes	625n	1.0
See Also	GO, K, KDRIVE, LHAD, LHADA, S	6270	1.0

The Kill Motion (<ctrl>K) command instructs the controller to instantaneously stop motion on all axes, and terminate program execution. In essence, the <ctrl>K command is an immediate kill (!K) command.

STEPPER SYSTEMS

This command should be used with caution. Since motion is stopped instantaneously, without a controlled deceleration ramp, high inertial loads may cause a drive to fault. A drive fault condition will allow the load to *free wheel*, possibly damaging equipment. Compumotor recommends using a brake on your motor drive system to brake the load in the event of a drive fault.

Servos: Motion is stopped at the rate set with the LHADA and LHAD commands. If the *Disable Dive on Kill* mode is enabled with the KDRIVE command, a <ctrl>K command immediately shuts down the drive, allowing the load to *free wheel* to a stop.

Example:

```
A2,2,25000,25000  ; Sets acceleration to 2, 2, 25000, and 25000 units/sec/sec
                  ; for axes 1, 2, 3 and 4
AD2,2,25000,25000 ; Sets deceleration to 2, 2, 25000, and 25000 units/sec/sec
                  ; for axes 1, 2, 3 and 4
V1,1,1,2          ; Sets velocity to 1, 1, 1, and 2 units/sec for axes 1, 2, 3
                  ; & 4, respectively
@D10              ; Set distance on all axes to 10 units
@GO1              ; Initiate motion on all axes -- motion begins.
                  ; After a short period the Kill command is sent.
<CTRL>K           ; Kill motion on all axes (steppers stop instantaneously,
                  ; servos stop at the LHADA/LHAD deceleration)
```

KDRIVE Disable Drive on Kill

		Product	Rev
Type	Controller Configuration	AT6n00	n/a
Syntax	<!><@><a>KDRIVE<b><b<b><b>	AT6n50	1.0
Units	b = enable bit	610n	n/a
Range	0 (disable), 1 (enable), or X (don't change)	615n	1.0
Default	0	620n	n/a
Response	KDRIVE: *KDRIVE0000 1KDRIVE: *KDRIVE0	625n	1.0
See Also	DRIVE, INFNC, K, <ctrl>K	6270	1.0

If you enable the Disable Drive on Kill function (KDRIVE1), then when a kill command (K, !K, or <ctrl>K) is processed or a kill input (INFNCi-C) is activated, the drive will be disabled immediately; this cuts all control to the motor and allows the load to freewheel to a stop.

When the drive is disabled (shutdown/de-energized) the SHTNO relay output is disconnected from COM, and the SHTNC relay output is connected to COM. To re-enable the drive, issue the DRIVE11 command.

If you leave the KDRIVE command in its default state (Ø, disabled), the kill function behaves in its normal manner, leaving the drive enabled.

Example:

```
KDRIVE11        ; Set axes 1 & 2 to de-energize the drive during a kill
K               ; Kill is performed and drives are de-energized
```

L — Loop

		Product	Rev
Type	Loops or Program Flow Control	AT6n00	1.0
Syntax	<!>L<i>	AT6n50	1.0
Units	i = number of times to loop	610n	4.0
Range	0 - 999,999,999	615n	1.0
Default	0	620n	1.0
Response	L: No response, instead this does the same thing as L0	625n	1.0
See Also	LN, LX, PLN, PLOOP	6270	1.0

When you combine the Loop (L) command with the end of loop (LN) command, all of the commands between L and LN will be repeated the number of times indicated by n. If <i> = Ø, or if no argument is specified, all the commands between L and LN will be repeated indefinitely. The loop can be stopped by issuing a Terminate Loop (!LX) command, an immediate Kill (!K) command, or an immediate Halt (!HALT) command.

The loop can be paused by issuing an immediate Pause (!PS) command or a Stop (!S) command with COMEXS enabled. The loop can then be resumed with the immediate Continue (!C) command. You may nest loops up to 16 levels deep.

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

Example:

```
L5          ; Repeat the commands between L and LN five times
GO1110      ; Start motion on axes 1, 2, and 3, axis 4 will remain motionless
LN          ; End loop
```

[LDT] — Position of LDT

		Product	Rev
Type	Assignment or Comparison	AT6n00	n/a
Syntax	See below	AT6n50	n/a
Units	See below	610n	n/a
Range	See below	615n	n/a
Default	n/a	620n	n/a
Response	n/a	625n	n/a
See Also	[AS], CMDDIR, ERROR, [FB], LDTPOL, PSET, SCALE, SCLD, SFB, TAS, TER, TFB	6270	1.0

Use the LDT command to assign the current value of the specified LDT to a variable or to make a comparison.

If scaling is not enabled (SCALEØ), the value will represent actual LDT counts. If scaling is enabled (SCALE1), the value will be scaled by the distance scaling factor (SCLD).

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

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

An LDT position read error can be caused by a bad LDT connection, an LDT failure, or an LDTUPD command value being too small. If this error occurs, axis status bit #27 (reported with the TAS and AS commands) will be set. In addition, if ERROR bit #15 is enabled (ERROR.15-1), error status bit #15 (reported with the TER and ER commands) will also be set.

Example:

```
VAR6=2LDT       ; Assign position of LDT #2 to variable #6
IF(1LDT<500)    ; If position of LDT #1 is less than 500, do IF commands
VAR4=1FB+1000   ; Set variable #4 equal to current position of LDT #1 plus 1,000
NIF             ; End of IF statement
```

LDTGRD LDT Gradient

Type	LDT Configuration
Syntax	<!><@><a>LDTGRD<r>,<r>
Units	r = gradient in µs/inch
Range	r = 3.0000 - 20.0000
Default	9.0000
Response	LDTGRD *LDTGRD9.0000,9.0000 1LDTGRD *1LDTGRD9.0000
See Also	DRIVE, [LDT], LDTRES, LDTUPD, SCLA, SCLD, SCLV, TLDT

Product	Rev
AT6n00	n/a
AT6n50	n/a
610n	n/a
615n	n/a
620n	n/a
625n	n/a
6270	1.0

Use the LDTGRD command to set the linear displacement transducer (LDT) gradient to calibrate the LDT feedback. The *gradient* is a measure of how quickly the LDT can respond to feedback requests. It is unique to each LDT and should be printed on the unit. The gradient is used to correct for positional differences created by different LDTs; this allows programs to be easily transported between 6270s or used with a new LDT.

The LDTGRD value may be changed only if the drive/value is disabled with the DRIVE command.

The LDTGRD setting is saved in non-volatile memory.

If the manufacturer of your LDT expresses the gradient in units other than µs/inch, convert the units to µs/inch so that you can enter an accurate gradient with the LDTGRD command.

The 9.000 µs/inch gradient provides a scale of 432 steps per inch. If the LDT gradient is other than 9 µs/inch it will be scaled internally to provide uniformity.

The LDT's position can be monitored with the TLDT and [LDT] commands.

Example:

```
LDTGRD9.0990,9.037  ; Set gradients LDTs on axes 1 and 2
```

LDTPOL LDT Polarity

Type	LDT; Controller Configuration
Syntax	<!><@><a>LDTPOL<b><b>
Units	b = polarity bit
Range	0 (normal polarity), 1 (reverse polarity) or X (don't change)
Default	0
Response	LDTPOL *LDTPOL00 1AINPOL *1LDTPOL0
See Also	CMDDIR, [FB], FOLMAS, [LDT], [PCL], [PE], PSET, SFB, TFB, TLDT, TPCL, TPER

Product	Rev
AT6n00	n/a
AT6n50	n/a
610n	n/a
615n	n/a
620n	n/a
625n	n/a
6270	3.0

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

If the LDT input is counting in the wrong direction, you may reverse the polarity with the LDTPOL command (see programming example below). This allows you to reverse the counting direction without having to change the actual mounting of the LDT device. For example, if the LDT on axis 2 counted in the wrong direction, you could issue the LDTPOLx1 command to correct the polarity.

Immediately after issuing the LDTPOL command, the sign of the LDT counts (including all LDT position registers) is reversed. The polarity is immediately changed whether or not LDT feedback is currently selected with the SFB command.

NOTE

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

The LDTPOL command is automatically saved in non-volatile RAM (stand-alone products only).

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

Example:

```
SFB1,3          ; Select encoder feedback for axis 1 and LDT feedback for axis 2
2TLDT           ; Example response is *2TLDT+1.294
                ; (response indicates LDT #2 is at position +1.294)
LDTPOLx1        ; Reverse LDT polarity on axis 2
2TLDT           ; Example response is *2TLDT-1.294
                ; (response indicates LDT #2 is at position -1.294)
```

LDTRES LDT Resolution

		Product	Rev
Type	LDT Configuration	AT6n00	n/a
Syntax	<!><@><a>LDTRES<i>,<i>	AT6n50	n/a
Units	i = counts per inch	610n	n/a
Range	i = 200 - 65535	615n	n/a
Default	432	620n	n/a
Response	LDTRES *LDTRES432,432 1LDTRES *1LDTRES432	625n	n/a
See Also	[AS], [ER], ERROR, ERES, LDTGRD, LDTUPD, SCALE, SCLD, SCLV, TAS, TER	6270	1.0

Use the LDTRES command to establish the LDT counts-per-inch resolution for programming the 6270. The 6270 counter frequency of 48 MHz provides a resolution of 432 counts/inch (assuming an LDT gradient of 9 µs/inch).

You can use the SCLA, SCLD, and SCLV commands to convert distance units from LDT counts to other, more convenient, units. To program in inches, use a scale factor of 432 (see example below). To program in millimeters use a scale factor of 17.

If recirculation is used, multiply 432 by the number of recirculations. (A *recirculation* is a single request for position information from the LDT. Multiple recirculations provide greater resolution—but reduced accuracy— by increasing the length of the feedback pulse. The number of recirculations is determined by the LDT and not the 6270. LDTs must be ordered from the manufacturer or be configured by the user for multiple recirculations.) For example, if you are using LDTs with 4 recirculations and you want to program in inches, you should set the resolution of each LDT to 1728 (4 ∗ 432 = 1728) with the LDTRES1728,1728 command.

If using recirculation, the time required to obtain a position reading is increased. If the LDT response time is too long, the 6270 will report a read error. If a read error occurs, axis status bit #27 (reported with the TAS and AS commands) will be set. In addition, if ERROR bit #15 is enabled (ERROR.15-1), error status bit #15 (reported with the TER and ER commands) will also be set. For more information, refer to the LDTUPD command.

Example

```
LDTRES864,864   ; Configure axes 1 & 2 for 2 recirculations
SCLA864,864     ; Program acceleration values in inches/sec/sec
SCLV864,864     ; Program velocity values in inches/sec
SCLD864,864     ; Program distance values in inches
SCALE1          ; Enable scaling
D5              ; Set move distance to 5 inches
GO11            ; Initiate motion on axes 1 and 2
```

LDTUPD LDT Position Update Rate

		Product	Rev
Type	LDT Configuration	AT6n00	n/a
Syntax	<!><@><a>LDTUPD<i>,<i>	AT6n50	n/a
Units	i = multiples of the system update rate	610n	n/a
Range	i = 1 - 1000	615n	n/a
Default	1	620n	n/a
Response	LDTUPD *LDTUPD1,1 1LDTUPD *1LDTUPD1	625n	n/a
See Also	[AS], [ER], ERROR, INDAX, LDTRES, LDTGRD, SSFR, TAS, TER	6270	1.0

The LDTUPD command value is multiplied by the *system update rate* to establish the LDT position sample rate. The system update rate is determined by the current SSFR and INDAX command settings (see table in SSFR command description).

As the LDTUPD value is increased (update rate is decreased), the quality of the dynamic response degrades. However, if the update rate is too fast, the LDT will not have enough time to read the position and LDT read errors will occur. If a read error occurs, axis status bit #27 (reported with the TAS and AS commands) will be set. In addition, if ERROR bit #15 is enabled (ERROR.15-1), error status bit #15 (reported with the TER and ER commands) will also be set.

To determine the minimum allowable LDTUPD update rate, use this formula:

((max. length of travel in inches ∗ LDT gradient) ∗ number of recirculations) + 140µs < LDTUPD value ∗ system update rate

Example:
```
LDTUPD6,6       ; Sample the LDT position once every 6 system update periods
```

LH Hard Limit Enable

		Product	Rev
Type	Limit (End-of-Travel)	AT6n00	1.0
Syntax	<!><@><a>LH<i>,<i>,<i>,<i>	AT6n50	1.0
Units	n/a	610n	4.0
Range	i = 0 (disable both), 1 (disable positive-direction), 2 (disable negative-direction), or 3 (enable both)	615n	1.0
Default	3	620n	1.0
Response	LH: *LH3,3,3,3 1LH: *1LH3	625n	1.0
See Also	[AS], [ER], ERROR, LHAD, LHADA, LHLVL, [LIM], LS, LSAD, LSADA, LSNEG, LSPOS, TAS, TER, TLIM, TSTAT	6270	1.0

The Hard Limit Enable (LH) command determines the status of the hard wired limits. With limits disabled, motion will not be restricted. When a specific limit is enabled (positive- or negative-direction), and the limit wiring for the enabled limit is a physical open circuit, motion will be restricted (assuming LHLVLØ). The active level of the limit inputs can be changed with the LHLVL command.

Disable negative- and positive-direction limits:	i = 0
Enable negative-direction, disable positive-direction limit:	i = 1
Enable positive-direction, disable negative-direction limit:	i = 2
Enable negative- and positive-direction limits:	i = 3

NOTE

If a hard limit is encountered while limits are enabled, motion must occur in the opposite direction after correcting the limit condition (resetting the switch); then you can make a move in the original direction. If limits are disabled, you are free to make a move in either direction.

Example:
```
LH3,3           ; Enable limits on axes 1 and 2
LHAD100,100     ; Set hard limit decel to 100 units/sec/sec on axes 1 and 2
LHLVL0000       ; Active low hard limit
A10,12          ; Set acceleration to 10 and 12 units/sec/sec for axes 1 and 2
V1,1            ; Set velocity to 1 unit/sec for axes 1 and 2
D100000,1000    ; Set distance to 100000 and 1000 units for axes 1 and 2
GO11XX          ; Initiate motion on axes 1 and 2
```

LHAD — Hard Limit Deceleration

Type	Limit (End-of-Travel)
Syntax	<!><@><a>LHAD<r>,<r>,<r>,<r>
Units	r = units/sec/sec
Range	0.00025 - 24,999,999 (depending on the scaling factor)
Default	100.0000
Response	LHAD: *LHAD100.0000,100.0000,100.0000,100.0000 1LHAD: *1LHAD100.0000
See Also	DRES, DRFLVL, K, LH, LHADA, LHLVL, [LIM], LS, LSAD, LSADA, LSNEG, LSPOS, SCALE, SCLA

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Hard Limit Deceleration (LHAD) command determines the value at which to decelerate after an end-of-travel limit has been hit.

Steppers: The entered value is always in reference to motor steps, not encoder steps, regardless of the ENC command setting. If scaling is not enabled (SCALEØ), the deceleration value is entered in motor revs/sec/sec; this value is internally multiplied by the drive resolution (DRES) value to obtain an deceleration value in motor steps/sec/sec for the motion trajectory calculations. If scaling is enabled (SCALE1), the entered deceleration value is internally multiplied by the deceleration scaling factor (SCLA) to convert user units/sec/sec to motor steps/sec/sec.

Servos: If scaling is not enabled (SCALEØ), the deceleration value is entered in encoder or resolver revs/sec/sec, LDT inches/sec/sec, or ANI volts/sec/sec; encoder or resolver and LDT values are internally multiplied by the encoder resolution (ERES) value or the LDT resolution (LDTRES) value to obtain an acceleration value in steps/sec/sec for the motion trajectory calculations. If scaling is enabled (SCALE1), the entered deceleration value is internally multiplied by the deceleration scaling factor (SCLA) to convert user units/sec/sec to encoder, resolver, LDT, or ANI steps/sec/sec.

When a drive fault, a Kill command (K, !K, or ^K), or a Kill input (INFNCi-C) occurs, motion is stopped at the rate set with the LHAD and LHADA commands. If the *Disable Drive on Kill* mode is enabled (KDRIVE1), the drive is immediately shut down upon a Kill command or input and allows the motor/load to *freewheel* to a stop without a controlled deceleration.

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

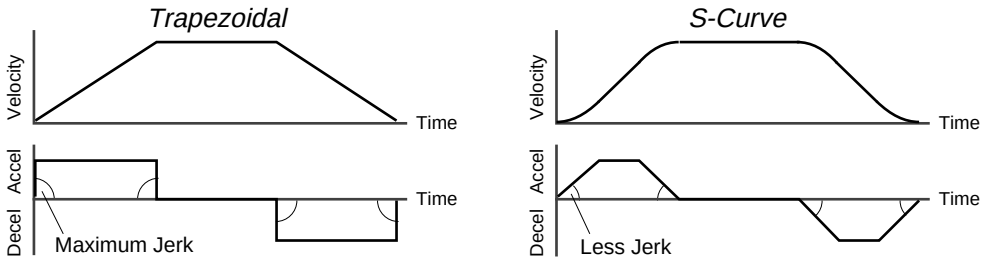
Example: Refer to the hard limit enable (LH) command example.

LHADA — Hard Limit Average Deceleration

Type	Motion (S-Curve)
Syntax	<!><@><a>LHADA<r>,<r>,<r>,<r>
Units	r = units/sec/sec
Range	0.00025 - 24999999 (depending on the scaling factor)
Default	100.000 (default is a constant deceleration ramp, where LHADA tracks LHAD)
Response	LHADA: *LHADA100.0000,100.000,100.000,100.000 1LHADA: *1LHADA100.0000,100.000
See Also	AD, ADA, K, LHAD, LHLVL, SCALE, SCLA

Product	Rev
AT6n00	n/a
AT6n50	1.0
610n	n/a
615n	1.0
620n	n/a
625n	1.0
6270	1.0

The Hard Limit Average Deceleration (LHADA) command allows you to specify the average deceleration for an S-curve deceleration profile when a limit is hit. S-curve profiling provides smoother motion control by reducing the rate of change in deceleration; this decel rate of change is known as *jerk*.



The values for the maximum hard limit decel (LHAD) and average hard limit decel (LHADA) commands determine the characteristics of the S-curve. To smooth the deceleration ramp, you must enter a LHADA command value that satisfies this equation: 1/2 LHAD ≤ LHADA < LHAD. The following conditions are possible:

Deceleration Setting	Profiling Condition
LHADA > 1/2 LHAD, but LHADA < LHAD	S-curve deceleration profile with a variable period of constant deceleration
LHADA = 1/2 LHAD	*Pure S-curve* (no period of constant deceleration—smoothest motion)
LHADA = LHAD	Trapezoidal profile (but can be changed to S-curve by specifying a new LHADA value < LHAD)
LHADA < 1/2 LHAD; or LHADA > LHAD	When you issue the LHADA command, the error message *INVALID CONDITIONS FOR S_CURVE ACCELERATION–FIELD *n* will be displayed.
AA = zero; or if no LHADA value is ever entered	Deceleration profile defaults to trapezoidal; LHADA will match the LHAD command value and will continue to match whatever value LHAD is set to.

NOTE

Once you enter an LHADA value that is ≠ zero or ≠ LHAD, S-curve profiling is enabled **only for hard limit move decelerations** (e.g., not for contouring decelerations, which require the PADA command).

Increasing the LHADA value above the pure S-curve level (LHADA > 1/2 LHAD), the time required to reach zero velocity decreases. However, increasing LHADA also increases jerk.

The calculation for determining S-curve average deceleration move times is as follows:

(A_{avg} = average deceleration value):

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

NOTE: Acceleration scaling (SCLA) affects LHADA the same as it does for LHAD.

*** For a more in-depth discussion on S-curve profiling, refer to the Custom Profiling chapter in the *6000 Series Programmer's Guide*.

Example:

```
LHAD10,10,10,10   ; Set the maximum deceleration of all four axes
LHADA5,5,7.5,10   ; Set the average deceleration of all four axes
```

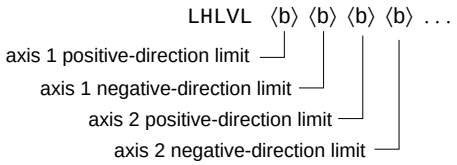
LHLVL — Hard Limit Active Level

Type	Limit (End-of-Travel)
Syntax	<!>LHLVL<b><b><b><b><b><b><b><b>
Units	n/a
Range	b = 0 (active low for a n.c. switch), 1 (active high for a n.o. switch), or X (don't change)
Default	0
Response	LHLVL: *LHLVL0000_0000
See Also	LH, LHAD, LHADA, [LIM], LS, LSAD, LSADA, LSNEG, LSPOS, TAS, TLIM

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Hard Limit Active Level (LHLVL) command defines the active state of all end-of-travel hard limit inputs. The default state is active low.

Command Format:



The end-of-travel limit switch can be either normally-open or normally-closed. If a normally-open switch is used, the hard limit active level should be set to active high (LHLVL1). If a normally-closed switch is used, the hard limit active level should be set to active low (LHLVLØ). Compumotor recommends that all end-of-travel limit switches be normally-closed, because with normally-closed limit switches, the limit function (i.e., inhibiting motion) is considered active when the switch contact is opened or if the wiring to the switch is broken.

The end-of-travel limit input schematic is shown in the 6000 Series product's *Installation Guide*.

The state of the limit inputs can be displayed with the TLIM command. The state of the limit function (i.e., inhibiting motion) can be displayed with the TAS command.

Example: Refer to the hard limit enable (LH) command example.

[LIM] — Limit Status

Type	Assignment or Comparison
Syntax	See below
Units	n/a
Range	n/a
Default	n/a
Response	n/a
See Also	LH, TLIM

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

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

Syntax: VARBn=LIM where n is the binary variable number,
or [LIM] can be used in an expression such as IF(LIM=b1XX1), or IF(LIM=h7)

There are 3 limit inputs per axis, home limit, positive-direction, and negative-direction end-of-travel limits. Each is available for assignment or comparison. If it is desired to assign only one limit input value to a binary variable, instead of the status of all the limit inputs, the bit select (.) operator can be used. The bit

select, in conjunction with the limit input number, is used to specify a specific limit input. For example, `VARB1=LIM.4` assigns limit input 4 to binary variable 1.

Format for binary assignment: `bbbbbbbbbbbb`
^ Bit #1 ^ Bit #12

Bit	Function	
1	Axis 1 - positive-direction Limit	
2	Axis 1 - negative-direction Limit	
3	Axis 1 - Home Limit	
4	Axis 2 - positive-direction Limit	
5	Axis 2 - negative-direction Limit	
6	Axis 2 - Home Limit	
7	Axis 3 - Positive-direction Limit	(AT6400 and AT6450)
8	Axis 3 - Negative-direction Limit	(AT6400 and AT6450)
9	Axis 3 - Home Limit	(AT6400 and AT6450)
10	Axis 4 - Positive-direction Limit	(AT6400 and AT6450)
11	Axis 4 - Negative-direction Limit	(AT6400 and AT6450)
12	Axis 4 - Home Limit	(AT6400 and AT6450)

Example:
```
IF(LIM=b11X1)   ; If both limit inputs on axis 1 and the positive-direction
                ; limit input on axis 2 are active, then do the statements
                ; between the IF and NIF
TLIM            ; Transfer limit status
NIF             ; End IF statement
```

LN — End of Loop

Type	Loops or Program Flow Control
Syntax	<!>LN
Units	n/a
Range	n/a
Default	n/a
Response	No response; used in conjunction with the L command
See Also	L, LX

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The End of Loop (`LN`) command marks the end of a loop. You must use this command in conjunction with the Loop (`L`) command. All buffered commands that you enter between the `L` and `LN` commands are executed as many times as the number that you enter following the `L` command. You may nest loops up to 16 levels deep. **NOTE**: Be careful about performing a `GOTO` between the `L` and `LN` commands. Branching to a different location within the same program will cause the next loop encountered to be nested within the previous loop, unless an `LN` command has already been encountered.

Example:
```
L5          ; Repeat the commands between L and LN five times
GO1110      ; Start motion on axes 1, 2, and 3, axis 4 will remain motionless
LN          ; End loop
```

LS — Soft Limit Enable

Type	Limit (End-of-Travel)
Syntax	<!><@><a>LS<i>,<i>,<i>,<i>
Units	n/a
Range	i = 0 (disable both), 1 (disable positive-direction), 2 (disable negative-direction) or 3 (Enable both)
Default	0
Response	LS: *LS0,0,0,0 1LS: *1LS0
See Also	[AS], [ER], LH, LHAD, LHADA, LHLVL, LSAD, LSADA, LSNEG, LSPOS, TAS, TER, TSTAT

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Soft Limit Enable (LS) command determines the status of the programmable soft move distance limits. With soft limits disabled, motion will not be restricted. After a soft limit absolute position has been programmed (LSPOS and LSNEG), and the soft limit is enabled (LS), a move will be restricted upon reaching the programmed soft limit absolute position. The rate at which motion is decelerated to a stop upon reaching a soft limit is determined by the LSAD and LSADA commands.

Disable negative- and positive-direction soft limits	i = 0
Enable negative-direction, disable positive-direction soft limit	i = 1
Enable positive-direction, disable negative-direction soft limit	i = 2
Enable negative- and positive-direction soft limits	i = 3

NOTE: The controller maintains an absolute count, even though you may be programming in the incremental mode (MAØ). The soft limits will also function in incremental mode (MAØ) or continuous mode (MC1). The soft limit position references the commanded position, not the position as measured by the feedback device (e.g., encoder).

NOTE

If a soft limit is encountered while limits are enabled, motion must occur in the opposite direction before a move in the original direction is allowed. You cannot use the PSET command to clear the soft limit condition. If limits are disabled, you are free to make a move in either direction.

Example:

```
SCALE0                ; Disable scaling.
LSPOS500000,50000     ; Set soft limit positive-direction absolute positions to be
                      ; 500000 units for axis 1, 50000 units for axis 2
                      ; (Soft limits are always absolute)
LSNEG-500000,-50000 ; Set soft limit negative-direction absolute positions to
                      ; be -500000 units for axis 1, -50000 units for axis 2
                      ; (Soft limits are always absolute)
LS3,3                 ; Soft limits are enabled on axes 1 and 2
LSAD100,100           ; Soft limit decel set to 100 units/sec/sec on axes 1 and 2
PSET0,0,0,0           ; Set absolute position on all axes to 0
A10,12                ; Set accel to 10 and 12 units/sec/sec for axes 1 and 2
V1,1                  ; Set velocity to 1 unit/sec for axes 1 and 2
D100000,1000          ; Set distance to 100000 and 1000 units for axes 1 and 2
GO11XX                ; Initiate motion on axes 1 and 2
```

LSAD — Soft Limit Deceleration

Type	Limit (End-of-Travel)
Syntax	<!><@><a>LSAD<r>,<r>,<r>,<r>
Units	r = units/sec/sec
Range	0.00025 - 24,999,999 (depending on the scaling factor)
Default	100.0000
Response	LSAD: *LSAD100.0000,100.0000,100.0000,100.0000 1LSAD: *1LSAD100.0000
See Also	DRES, LH, LHAD, LHADA, LHLVL, LS, LSADA, LSNEG, LSPOS, SCALE, SCLA

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Soft Limit Deceleration (LSAD) command determines the value at which to decelerate after a programmed soft limit (LSPOS or LSNEG) has been hit.

Steppers: The entered value is always in reference to motor steps, not encoder steps, regardless of the ENC command setting. If scaling is not enabled (SCALEØ), the deceleration value is entered in motor revs/sec/sec; this value is internally multiplied by the drive resolution (DRES) value to obtain an deceleration value in motor steps/sec/sec for the motion trajectory calculations. If scaling is enabled (SCALE1), the entered deceleration value is internally multiplied by the deceleration scaling factor (SCLA) to convert user units/sec/sec to motor steps/sec/sec.

Servos: If scaling is not enabled (SCALEØ), the deceleration value is entered in encoder or resolver revs/sec/sec, LDT inches/sec/sec, or ANI volts/sec/sec; encoder or resolver and LDT values are internally multiplied by the encoder resolution (ERES) value or the LDT resolution (LDTRES) value to obtain an acceleration value in steps/sec/sec for the motion trajectory calculations. If scaling is enabled (SCALE1), the entered deceleration value is internally multiplied by the deceleration scaling factor (SCLA) to convert user units/sec/sec to encoder, resolver, LDT, or ANI steps/sec/sec.

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

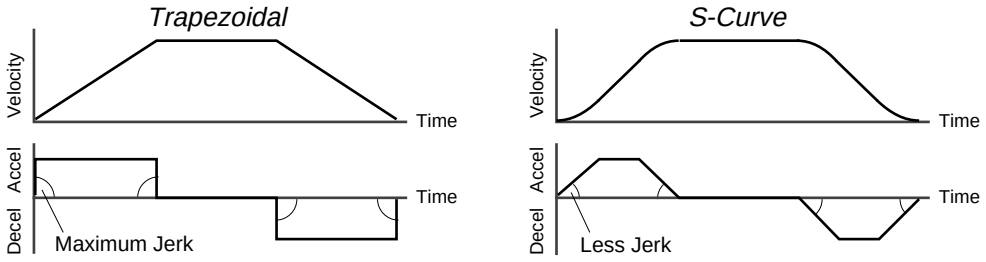
Example: Refer to the soft limit enable (LS) command example.

LSADA Soft Limit Average Deceleration

Type	Motion (S-Curve)
Syntax	<!><@><a>LSADA<r>,<r>,<r>,<r>
Units	r = units/sec/sec
Range	0.00025 - 24999999 (depending on the scaling factor)
Default	100.0000 (default is a constant deceleration ramp, where LSADA tracks LSAD)
Response	LSADA: *LSADA100.0000,100.000,100.000,100.000 1LSADA: *1LSADA100.0000
See Also	AD, ADA, LS, LSAD, SCALE, SCLA

Product	Rev
AT6n00	n/a
AT6n50	1.0
610n	n/a
615n	1.0
620n	n/a
625n	1.0
6270	1.0

The Soft Limit Average Deceleration (LSADA) command allows you to specify the average deceleration for an S-curve deceleration profile when a soft limit is hit. S-curve profiling provides smoother motion control by reducing the rate of change in deceleration; this decel rate of change is known as *jerk*.



The values for the maximum soft limit decel (LSAD) and average soft limit decel (LSADA) commands determine the characteristics of the S-curve. To smooth the deceleration ramp, you must enter a LSADA command value that satisfies this equation: 1/2 LSAD ≤ LSADA < LSAD. The following conditions are possible:

Deceleration Setting	Profiling Condition
LSADA > 1/2 LSAD, but LSADA < LSAD	S-curve deceleration profile with a variable period of constant deceleration
LSADA = 1/2 LSAD	*Pure S-curve* (no period of constant deceleration—smoothest motion)
LSADA = LSAD	Trapezoidal profile (but can be changed to S-curve by specifying a new LSADA value < LSAD)
LSADA < 1/2 LSAD; or LSADA > LSAD	When you issue the LSADA command, the error message *INVALID CONDITIONS FOR S_CURVE ACCELERATION–FIELD *n* will be displayed.
AA = zero; or if no LSADA value is ever entered	Deceleration profile defaults to trapezoidal; LSADA will match the LSAD command value and will continue to match whatever value LSAD is set to.

NOTE

Once you enter an LSADA value that is ≠ zero or ≠ LSAD, S-curve profiling is enabled **only for the soft limit move deceleration** (e.g., not for the hard limit deceleration, which requires the LHADA command).

Increasing the LSADA value above the pure S-curve level (LSADA > 1/2 LSAD), the time required to reach zero velocity decreases. However, increasing LSADA also increases jerk.

The calculation for determining S-curve average deceleration move times is as follows:

(A_{avg} = average deceleration value):

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

NOTE: Acceleration scaling (SCLA) affects LSADA the same as it does for LSAD.

*** For a more in-depth discussion on S-curve profiling, refer to the Custom Profiling chapter in the *6000 Series Programmer's Guide*.

Example:

```
LSAD10,10,10,10   ; Sets the maximum deceleration of all four axes
LSADA5,5,7.5,10   ; Sets the average deceleration of all four axes
```

LSNEG Soft Limit Negative Travel Range

		Product	Rev
Type	Limit (End-of-Travel)	AT6n00	1.0
Syntax	<!><@><a>LSNEG<r>,<r>,<r>,<r>	AT6n50	1.0
Units	r = units of distance	610n	4.0
Range	-999,999,999 - +999,999,999 (scalable)	615n	1.0
Default	+0	620n	1.0
Response	LSNEG: *LSNEG+0,+0,+0,+0 1LSNEG: *1LSNEG+0	625n	1.0
See Also	LH, LHAD, LHADA, LHLVL, LS, LSAD, LSADA, LSPOS, PSET, SCALE, SCLD	6270	1.0

The LSNEG command specifies the distance in absolute units where motion will be restricted when traveling in a negative-travel direction. The reference position used to determine absolute position is set to zero upon power-up, and can be reset using the PSET command. **Be sure to set the LSPOS value greater than the LSNEG value.**

The LSNEG value remains set until you change it with a subsequent LSNEG command.

All soft limit values entered are in absolute steps. If scaling is enabled (SCALE1), LSNEG is internally multiplied by the distance scale factor (SCLD). The soft limit position references the commanded position, not the position as measured by the feedback device (e.g., encoder).

Example: Refer to the soft limit enable (LS) command example.

LSPOS — Soft Limit Positive Travel Range

		Product	Rev
Type	Limit (End-of-Travel)	AT6n00	1.0
Syntax	<!><@><a>LSPOS<r>,<r>,<r>,<r>	AT6n50	1.0
Units	r = units of distance	610n	4.0
Range	-999,999,999 - +999,999,999 (scalable)	615n	1.0
Default	+0	620n	1.0
Response	LSPOS: *LSPOS+0,+0,+0,+0 1LSPOS: *1LSPOS+0	625n	1.0
See Also	LH, LHAD, LHADA, LHLVL, LS, LSAD, LSADA, LSNEG, PSET, SCALE, SCLD	6270	1.0

The LSPOS command specifies the distance in absolute units where motion will be restricted when traveling in a positive-travel direction. The reference position used to determine absolute position is set to zero upon power-up, and can be reset using the PSET command. **Be sure to set the LSPOS value greater than the LSNEG value.**

The LSPOS value remains set until you change it with a subsequent LSPOS command.

All soft limit values entered are in absolute steps. If scaling is enabled (SCALE1), LSPOS is internally multiplied by the distance scale factor (SCLD). The soft limit position references the commanded position, not the position as measured by the feedback device (e.g., encoder).

Example: Refer to the soft limit enable (LS) command example.

LX — Terminate Loop

		Product	Rev
Type	Loops or Program Flow Control	AT6n00	1.0
Syntax	<!>LX	AT6n50	1.0
Units	n/a	610n	4.0
Range	n/a	615n	1.0
Default	n/a	620n	1.0
Response	n/a	625n	1.0
See Also	L, LN, PLN, PLOOP	6270	1.0

The Terminate Loop (LX) command terminates the current loop in progress. This command does not halt processing of the commands in the loop until the last command in the current loop iteration is executed. At this point, the loop is terminated. If there are nested loops, only the inner most loop is terminated.

This command can be used externally to terminate the loop only if it is preceded by the immediate command specifier (!LX). If the immediate command specifier is not used, the command will have no effect on a loop in progress. An example of where the buffered Terminate Loop command (LX) might be used is provided below.

Example:

```
; ********************************************************************
; This program will make the move specified by the GO1110 command
; indefinitely until input 2 goes high, at which point, an LX will
; be issued, terminating the loop.
; ********************************************************************
;
L0              ; Repeat the commands between L and LN infinitely, or until
                ; a Terminate Loop (LX) command is received
GO1110          ; Start motion on axes 1, 2, and 3,
                ; axis 4 will remain motionless
IF(IN=bX1)      ; If input 2 goes high execute all statements between IF and NIF
LX              ; Terminate loop
NIF             ; End IF statement
LN              ; End loop
```

MA — Absolute/Incremental Mode Enable

Type	Motion
Syntax	<!><@><a>MA<b><b><b><b>
Units	n/a
Range	b = 0 (incremental mode) or 1 (absolute mode)
Default	0 (1 for 6270 only)
Response	MA: *MA0000 1MA: *1MA0
See Also	COMEXC, D, GO, GOBUF, PSET

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Absolute/Incremental Mode Enable (MA) command specifies whether the moves to follow are made with respect to current position (incremental) or with respect to an absolute zero position.

In incremental mode (MAØ), all moves are made with respect to the position at the beginning of the move. This mode is useful for repeating moves of the same distance.

In absolute mode (MA1), all moves are made with respect to the absolute zero position. The absolute zero position is equal to zero upon power up, and can be redefined with the PSET command. An internal counter keeps track of absolute position.

ON-THE-FLY CHANGES (as of revision 4.0): You can change positioning modes *on the fly* (while motion is in progress) in two ways. One way is to send an immediate command (!MA) followed by an immediate go command (!GO). The other way is to enable the continuous command execution mode (COMEXC1) and execute a buffered command (MA) followed by a buffered go command (GO).

Example:

```
PSET0,0,0,1000    ; Set absolute position on axes 1, 2, and 3 to zero,
                  ; and axis 4 to 1000 units
MA1111            ; Enable absolute mode on axes 1 through 4
A2,2,25000,25000  ; Set acceleration to 2, 2, 25000, and 25000 units/sec/sec
                  ; for axes 1, 2, 3 and 4
AD2,2,25000,25000 ; Set deceleration to 2, 2, 25000, and 25000 units/sec/sec
                  ; for axes 1, 2, 3 and 4
V1,1,1,2          ; Set velocity to 1, 1, 1, and 2 units/sec
                  ; for axes 1, 2, 3 and 4 respectively
@D10              ; Set distance on all axes to 10 units
GO1111            ; Initiate motion on all axes (axes 1, 2, and 3 will move
                  ; 10 units in the positive direction, axis 4 will move
                  ; 990 units in the negative direction)
```

MC — Preset/Continuous Mode Enable

Type	Motion
Syntax	<!><@><a>MC<b><b><b><b>
Units	n/a
Range	b = 0 (preset mode) or 1 (continuous mode)
Default	0
Response	MC: *MC0000 1MC: *1MC0
See Also	A, AD, COMEXC, COMEXS, D, FOLMD, [FS], FSHFC, FSHFD, GO, GOBUF, K, MA, PSET, S, SSV, TEST, TFS, V

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Preset/Continuous Mode Enable (MC) command causes subsequent moves to go a specified distance (MCØ), or a specified velocity (MC1).

In the Preset Mode (MCØ), all moves will go a specific distance. The actual distance traveled is specified by the D, SCLD, and MA commands.

In the Continuous Mode (MC1), all moves will go to a specific velocity with the Distance (D) command establishing the direction (D+ or D-). The actual velocity will be determined by the V and SCLV commands, or the V and DRES commands.

Motion will stop with an immediate Stop (!S) command, an immediate Kill (!K) command, or by specifying a velocity of zero followed by a GO command. Motion can also be stopped with a buffered Stop (S) or Kill (K) command if the continuous command processing mode (COMEXC) is enabled.

ON-THE-FLY CHANGES (as of revision 4.0): You can change positioning modes *on the fly* (while motion is in progress) in two ways. One way is to send an immediate command (!MC) followed by an immediate go command (!GO). The other way is to enable the continuous command execution mode (COMEXC1) and execute a buffered command (MC) followed by a buffered go command (GO).

Example:

```
MA0000              ; Enable incremental mode on all axes
MC0000              ; Enable preset mode on all axes
A2,2,25000,25000    ; Set acceleration to 2, 2, 25000, and 25000 units/sec/sec
                    ; for axes 1, 2, 3 & 4
AD2,2,25000,25000   ; Set deceleration to 2, 2, 25000, and 25000 units/sec/sec
                    ; for axes 1, 2, 3 & 4
V1,1,1,2            ; Set velocity to 1, 1, 1, and 2 units/sec for
                    ; axes 1, 2, 3 & 4 respectively
D10,10,10,10        ; Set distance on all axes to 10 units
GO1111              ; Initiate motion on all axes (axes 1,2, 3, & 4 will
                    ; all move 10 units positive-direction)
COMEXC1             ; Enable continuous command processing mode
MC1111              ; Enable continuous mode on all axes
A8,8,2000,2000      ; Set acceleration to 8, 8, 2000, and 2000 units/sec/sec for
                    ; axes 1, 2, 3 & 4
AD8,8,2000,2000     ; Set deceleration to 8, 8, 2000, and 2000 units/sec/sec for
                    ; axes 1, 2, 3 & 4
V5,5,5,9            ; Set velocity to 5, 5, 5, and 9 units/sec for axes 1, 2, 3 & 4
GO1111              ; Initiate motion on all axes (axes 1,2, and 3 will each
                    ; travel at a velocity of 1 unit/sec, axis 4 will travel
                    ; at a velocity of 2 units/sec)
T15                 ; Wait 15 seconds
@V5                 ; Set velocity to 5 units/sec (axis 4 only affected axis)
GO1111              ; Initiate motion with new velocity of 5 units/sec (all axes)
T8                  ; Wait 8 seconds
@V0                 ; Set velocity to zero
GO1111              ; Initiate motion with new velocity of 5 units/sec (all axes)
WAIT(MOV=b0000)     ; Wait for motion to come to a halt on all axes
COMEXC0             ; Disable continuous command processing mode
```

MEMORY Partition User Memory

Type	Controller Configuration
Syntax	<!>MEMORY<i>,<i>
Units	i = bytes of memory (use even number only) 1st <i> = partition for "Programs" 2nd <i> = partition for "Compiled Profiles"
Range	(see table below)
Default	(see table below)
Response	MEMORY: *MEMORY33000,31000
See Also	[DATP], DEF, GOBUF, PCOMP, [SEG], [SS], TDIR, TMEM, TSEG, TSS

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.5
625n	1.0
6270	1.0

Your controller's memory has two partitions: one for storing *programs* and one for storing *compiled profiles*. The allocation of memory to these two areas is controlled with the MEMORY command.

"Programs" vs. "Compiled Profiles"

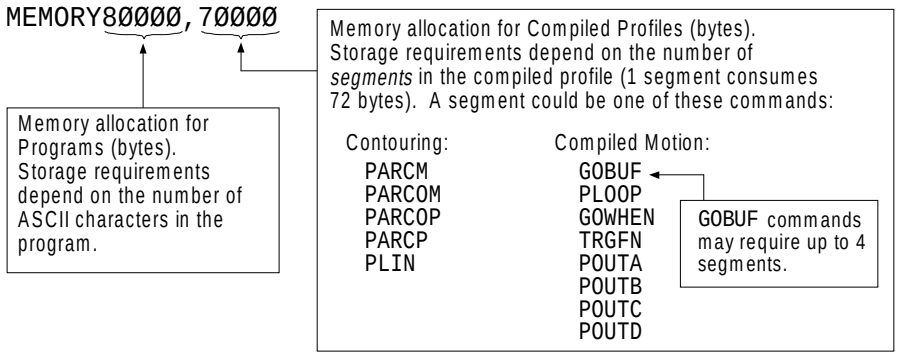
Programs are defined with the DEF and END commands, as demonstrated in the "Program Development Scenario" in the *6000 Series Programmer's Guide*.

Compiled Profiles are defined like programs (using the DEF and END commands), but are compiled with the PCOMP command and executed with the PRUN command. A compiled profile could be a multi-axis *contour* (a series of arcs and lines), an *individual axis profile* (a series of GOBUF commands), or a *compound profile* (combination of multi-axis contours and individual axis profiles).

Programs intended to be compiled are stored in program memory. After they are compiled with the PCOMP command, they remain in program memory and the *segments* (see diagram below) from the compiled profile are stored in compiled memory. The TDIR command reports which programs are compiled as a compiled profile (referred to here as a *path*).

For more information on multi-axis contours, refer to *Contouring* in the *6000 Series Programmer's Guide*. For more information on compiled profiles for individual axes, refer to *Compiled Motion Profiling* in the *6000 Series Programmer's Guide*.

MEMORY Syntax:



Allocation Defaults and Limits (by Product):

The following table identifies memory allocation defaults and limits for 6000 Series products. When specifying the memory allocation, use only even numbers. The minimum storage capacity for one partition area (program or compiled) is 1,000 bytes.

Feature	AT6n00	AT6n00-M	AT6n50	AT6n50-M	All Other Products
Total memory (bytes)	64000	1500000	40000	150000	150000
Default allocation (program,compiled)	33000,31000	63000,1000	39000,1000	149000,1000	149000,1000
Maximum allocation for programs	63000,1000	1499000,1000	39000,1000	149000,1000	149000,1000
Maximum allocation for compiled profiles	1000,63000	1000,1499000	1000,39000	1000,149000	1000,149000
Max. # of programs	150	4000	100	400	400
Max. # of labels	250	6000	100	600	600
Max. # of compiled profiles	100	800	100	300	300
Max. # of compiled profile segments	875	20819	541	2069	2069
Max. # of numeric variables	100	200	150	150	150
Max. # of string variables	100	100	25	25	25
Max. # of binary variables	100	100	25	25	25

-M refers to the Expanded Memory Option

When teaching variable data to a data program (DATP), be aware that the memory required for each data statement of four data points (43 bytes) is taken from the memory allocation for program storage.

CAUTION

Issuing a memory allocation command (e.g., MEMORY30000,10000) will erase all existing programs and compiled profile segments. However, issuing the MEMORY command by itself (e.g., type MEMORY <cr> by itself to request the status of how the memory is allocated) will not affect existing programs or compiled segments.

Checking Memory Status:

To find out what programs reside in your controller's memory, and how much of the available memory is allocated for programs and compiled profile segments, issue the TDIR command (see example response below). Entering the TMEM command or the MEMORY command (without parameters) will also report the available memory for programs and compiled profile segments.

Sample response to TDIR command:

```
*1 - SETUP USES 345 BYTES
*2 - PIKPRT USES 333 BYTES
*32322 OF 33000 BYTES (98%) PROGRAM MEMORY REMAINING
*500 OF 500 SEGMENTS (100%) COMPILED MEMORY REMAINING
```

Two system status bits (reported with the TSS, TSSF and SS commands) are available to check when compiled profile segment storage is 75% full or 100% full. System status bit #29 is set when segment storage reaches 75% of capacity; bit #30 indicates when segment storage is 100% full.

Example:

```
MEMORY56000,8000  ; Set aside 56000 bytes for program storage,
                  ; 8000 bytes for compiled profile segments
```

[MOV] Axis Moving Status

Type	Assignment or Comparison
Syntax	See below
Units	n/a
Range	n/a
Default	n/a
Response	n/a
See Also	[AS], GO, TAS

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

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

The axis moving status is also reported with bit #1 of the TAS, TASF and AS commands

Syntax: VARBn=MOV where n is the binary variable number,
or [MOV] can be used in an expression such as IF(MOV=b1XX1), or IF(MOV=h3)

Each bit of the MOV command corresponds to a specific axis. The first bit (left to right) is for axis 1, the second is for axis 2, etc. If the specific axis is in motion, the bit will be a one (1). If the specific axis is not in motion, the bit will be a zero (Ø).

Each 6000 Series product has 1 moving/not moving bit per axis. For example, the AT6400 has 4 axes, thus 4 moving/not moving bits. If it is desired to assign only one moving/not moving bit to a binary variable, instead of all the moving/not moving bits, the bit select (.) operator can be used. The bit select operator, in conjunction with the moving/not moving bit number, are used to specify a specific moving/not moving bit. For example, VARB1=MOV.2 assigns bit 2 (representing axis 2 moving/not moving) to binary variable 1.

Example:

```
COMEXC1            ; Enable continuous command processing mode
COMEXS1            ; Save command buffer on stop
MC1111             ; Enable continuous mode on all axes
A2,2,25000,25000   ; Set acceleration to 2, 2, 25000, and 25000 units/sec/sec
                   ; for axes 1, 2, 3 and 4 respectively
AD2,2,25000,25000  ; Set deceleration to 2, 2, 25000, and 25000 units/sec/sec
                   ; for axes 1, 2, 3 and 4 respectively
V1,1,1,2           ; Set velocity to 1, 1, 1, and 2 units/sec for axes 1, 2, 3
                   ; and 4 respectively
GO1111             ; Initiate motion on all axes (axes 1,2, and 3 will each
                   ; travel at a velocity of 1 unit/sec, axis 4 will travel
                   ; at a velocity of 2 units/sec)
T5                 ; Wait 5 seconds
S1111              ; Stop motion on all axes
WAIT(MOV=b0000)    ; Wait for motion to come to a halt on all axes
COMEXC0            ; Disable continuous command processing mode
```

NIF End IF Statement

Type	Program Flow Control or Conditional Branching
Syntax	<!>NIF
Units	n/a
Range	n/a
Default	n/a
Response	No response when used in conjunction with the IF command
See Also	ELSE, IF

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

This command is used in conjunction with the IF and ELSE commands to provide conditional program flow. If the expression contained within the parentheses of the IF command evaluates true, then the commands between the IF and the ELSE are executed. The commands between the ELSE and the NIF are

ignored. If the expression evaluates false, the commands between the ELSE and the NIF are executed. The commands between IF and ELSE are ignored. The ELSE command is optional and does not have to be included in the IF statement.

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

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

Example:
```
IF(IN=b1X0)     ; Specify IF condition to be input 1 = 1, input 3 = Ø
T5              ; IF condition evaluates true wait 5 seconds
ELSE            ; Else part of IF condition
TPE             ; IF condition does not evaluate true, transfer position
                ; of all encoders
NIF             ; End IF statement
```

[NMCY] Master Cycle Number

Type	Following; Assignment or Comparison
Syntax	See below
Units	n/a
Range	n/a
Default	n/a
Response	n/a
See Also	FMCLEN, FMCNEW, FMCP, [FS], [PMAS], TFS, TNMCY, TRGFN

Product	Rev
AT6n00	3.0
AT6n50	3.0
610n	4.0
615n	3.0
620n	3.0
625n	3.0
6270	3.0

The Master Cycle Number (NMCY) command is used to assign the current master cycle number (specific to one axis) to a numeric variable, or to make a comparison against another value. **The master must be assigned first (FOLMAS command) before this command will be useful.**

The value represents the current cycle number, not the position of the master (or the slave). The master cycle number is set to zero when master cycle counting is restarted, and is incremented each time a master cycle finishes (i.e., rollover occurs). It will often correspond to the number of complete parts in a production run. This value may be used for subsequent decision making, or simply recording the cycle number corresponding to some other event.

For a complete discussion of master cycles, refer to the Following chapter in the *6000 Series Programmer's Guide.*

Syntax: VARn=aNMCY where n is the variable number and a is the axis number, or NMCY can be used in an expression such as IF(1NMCY>=5). The NMCY command must be used with an axis specifier, or it will default to axis 1 (e.g., VAR1=1NMCY, IF(2NMCY>12), etc.).

Example:
```
IF(2NMCY>500)   ; If the master for axis 2 has moved through 500 cycles,
                ; do the IF statement
WRITE"500 cycles have occurred"  ; Send string to serial port or the AT-bus
NIF             ; End of IF statement
VAR12=3NMCY     ; Set VAR12 to equal the number of cycles that have
                ; occurred on axis 3 master
```

[NOT] Not

Type	Operator (Logical)
Syntax	See below
Units	n/a
Range	n/a
Default	n/a
Response	n/a
See Also	[AND], IF, NWHILE, [OR], REPEAT, UNTIL, WAIT, WHILE

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The NOT operator is used in conjunction with the program flow control commands (IF, REPEAT..UNTIL, WHILE..NWHILE, WAIT). The NOT operator compliments a logical expression. If an expression is true, the NOT operator will make the expression false. If an expression is false, the NOT operator will make the expression true. This fact is best illustrated by the following examples:

If variable #1 equals 1, then the following is a true statement: IF(VAR1<3)
By using the NOT operator, the same statement becomes false: IF (NOT VAR1<3)
If variable #2 equals 2, then the following statement is false: WHILE(VAR2=3)
By using the NOT operator, the same statement becomes true: WHILE (NOT VAR2=3)

To evaluate an expression (NOT Expression) to determine if the expression is true, use the following rule:

NOT TRUE = FALSE
NOT FALSE = TRUE

In the following example, variable #1 is displayed, then is incremented by 1 as long as VAR1 is not equal to 10.

Example:

```
VAR1=1              ; Set variable 1 equal to 1
WHILE(NOT VAR1=10) ; Compare variable 1 to 10, and logically not the expression
WRVAR1              ; Write out variable 1
VAR1=VAR1 + 1       ; Set variable 1 to increment 1 by 1
NWHILE              ; End WHILE statement
```

NWHILE End WHILE Statement

Type	Program Flow Control or Conditional Branching
Syntax	<!>NWHILE
Units	n/a
Range	n/a
Default	n/a
Response	No response when used in conjunction with the WHILE command
See Also	WHILE

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The WHILE command, in conjunction with the NWHILE command, provide a means of conditional program flow. The WHILE command marks the beginning of the conditional statement, the NWHILE command marks the end. If the expression contained within the parenthesis of the WHILE command evaluates true, then the commands between the WHILE and NWHILE are executed, and continue to execute as long as the expression evaluates true. If the expression evaluates false, then program execution jumps to the first command after the NWHILE.

Up to 16 levels of WHILE .... NWHILE commands may be nested.

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

Programming order: WHILE(expression) ...commands... NWHILE

Example:

```
WHILE(IN=b1X0)  ; While input 1 = 1, input 3 = Ø, execute commands between
                ; WHILE and NWHILE
T5              ; Wait 5 seconds
TPE             ; Transfer position of all encoders
NWHILE          ; End WHILE statement
```

ONCOND On Condition Enable

		Product	Rev
Type	On Condition (Program Interrupt)	AT6n00	1.0
Syntax	<!><@>ONCOND<b><b><b><b>	AT6n50	1.0
Units	n/a	610n	4.0
Range	b = 0 (disable), 1 (enable) or X (don't change)	615n	1.0
Default	0	620n	1.0
Response	ONCOND: *ONCOND0000	625n	1.0
See Also	FSHFD, ONIN, ONP, ONUS, ONVARA, ONVARB, [SS], TSS	6270	1.0

The On Condition Enable (ONCOND) command enables the ONIN, ONUS, ONVARA, and ONVARB commands. When enabled, the expressions specified in the ONIN, ONUS, ONVARA, and ONVARB commands will be continuously evaluated. If any of the expressions ever evaluate true, a GOSUB will be made to the ONP program/subroutine.

ONP, ONIN, ONUS, ONVARA, and ONVARB should be defined before enabling the On Condition. If ONP is not defined first, the error message *UNDEFINED LABEL will appear.

ONCONDbbbb: First b = ONIN Enable
Second b = ONUS Enable
Third b = ONVARA Enable
Fourth b = ONVARB Enable

When ON conditions WILL NOT interrupt immediately: These are situations in which an ON condition does not immediately interrupt the program in progress. However, the fact that the ON condition evaluated true is retained, and when the condition listed below is no longer preventing the interrupt, the interrupt will occur.

- While a WAIT statement is in progress
- While a time delay (T) is in progress
- While a program is being defined (DEF)
- While a pause (PS) is in progress
- While a data read (DREAD, DREADF, or READ) is in progress
- While motion is in progress due to GO, GOL, GOWHEN, HOM, JOY, JOG, or PRUN and the continuous command execution mode is disabled (COMEXCØ).

Example:

```
DEF bigmov      ; Define program bigmov
D20,20,1,3      ; Sets move distance on axes 1 and 2 to 20 units,
                ; axis 3 to 1 unit, and axis 4 to 3 units
GO1111          ; Initiate motion on all axes
END             ; End program definition
ONP bigmov      ; Set ON program to bigmov
ONINxxx1        ; On input #4 GOSUB to ONP program
ONCOND1000      ; Enable ONIN condition
;
;  Now that the ONP program named bigmov is defined, if input #4 becomes
;  active during normal program operation, the program will GOSUB to the
;  ONP program (bigmov).
```

ONIN — On an Input Condition Gosub

Type	On Condition (Program Interrupt)
Syntax	<!>ONIN<b><b><b>...<b><b>
Units	n/a
Range	b = 0 (disable), 1 (enable) or X (don't care)
Default	0
Response	ONIN: *ONIN0000_0000_0000_0000_0000_0000_0000
See Also	INFNC, ONCOND, ONP, TIN

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The On an Input Condition Gosub (ONIN) command specifies the input bit pattern which will cause a branch to the ON program (ONP). If the input pattern occurs, a GOSUB is performed. The subroutine or program that the GOSUB branches to is selected with the ON program (ONP) command.

Input bit assignments for the ONIN command vary by product. The input bit patterns for all 6000 products are provided on page 6 of this document. The inputs are numbered 1 to *n* (*n* depends on the product) from left to right.

The ONIN command must be enabled using the ONCOND command before any branching will occur. Once a branch to the ONP program occurs, ONIN command will not call the ONP program while the ONP program is executing, eliminating the possibility of recursive calls. After returning from the ONP program, the input pattern specified by the ONIN command must evaluate false before another branch to the ONP program, resulting from the ONIN inputs, will be allowed.

Example:

```
DEF bigmov      ; Define program bigmov
D20,20,1,3      ; Sets move distance on axes 1 and 2 to 20 units,
                ; axis 3 to 1 unit, and axis 4 to 3 units
GO1111          ; Initiate motion on all axes
END             ; End program definition
ONP bigmov      ; Set ON program to bigmov
ONINxxx1        ; On input #4 GOSUB to ONP program
ONCOND1000      ; Enable ONIN condition
;
; Now that the ONP program named bigmov is defined, if input #4 becomes
; active during normal program operation, the program will GOSUB to
; the ONP program (bigmov).
```

ONP — On Condition Program Assignment

Type	On Condition (Program Interrupt)
Syntax	<!>ONP<t>
Units	t = text (name of On Condition program)
Range	text name of 6 characters or less
Default	n/a
Response	ONP: *ONP bigmov
See Also	DEF, END, ONCOND, ONIN, ONUS, ONVARA, ONVARB

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The On Condition Program (ONP) command assigns the program to which programming will GOSUB when an ON condition is met. The program must be defined (DEF) previous to the execution of the ONP command. The ONP command must be specified before enabling the ON conditions (ONCOND). If ONP is not defined first, the error message *UNDEFINED LABEL will appear.

To unassign the program as the ON condition program, issue the ONP CLR command. Deleting the program with the DEL command will accomplish the same thing.

Within the ONP program, the programmer is responsible for checking which ON condition caused the branch, if multiple ON conditions (ONCOND) have been enabled. Once a branch to the ONP program occurs, the ONP program will not be called again until after it has finished executing. After returning from the ONP program, the condition that caused the branch must evaluate false before another branch to the ONP program will be allowed.

Example:

```
DEF bigmov      ; Define program bigmov
D20,20,1,3      ; Sets move distance on axes 1 and 2 to 20 units,
                ; axis 3 to 1 unit, and axis 4 to 3 units
GO1111          ; Initiate motion on all axes
END             ; End program definition
ONP bigmov      ; Set ON program to bigmov
ONINxxx1        ; On input #4 GOSUB to ONP program
ONCOND1000      ; Enable ONIN condition
;
; Now that the ONP program named bigmov is defined, if input #4 becomes
; active during normal program operation, the program will GOSUB to
; the ONP program (bigmov).
```

ONUS — On a User Status Condition Gosub

Type	On Condition (Program Interrupt)
Syntax	<!>ONUS<b><b><b>...<b><b> (16 bits)
Units	n/a
Range	b = 0 (disable), 1 (enable) or X (don't care)
Default	0
Response	ONUS: *ONUS0000_0000_0000_0000
See Also	INDUSE, INDUST, ONCOND, ONP

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The On a User Status Condition Gosub (ONUS) command specifies the user status bit pattern, defined using the INDUST command, which will cause a branch to the ON program (ONP). If the bit pattern occurs, a GOSUB is performed. The subroutine or program that the GOSUB branches to is selected by the ON program (ONP) command.

The ONUS command must be enabled using the ONCOND command before any branching will occur. Once a branch to the ONP program occurs, ONUS command will not call the ONP program while the ONP program is executing, eliminating the possibility of recursive calls. After returning from the ONP program, the user status bit pattern specified by the ONUS command must evaluate false before another branch to the ONP program, resulting from the ONUS status bits, will be allowed.

Example:

```
INDUSE1         ; Enable user status
INDUST1-5A      ; User status bit 1 defined as axis 1 status bit 5
INDUST2-3D      ; User status bit 2 defined as axis 4 status bit 3
INDUST3-5J      ; User status bit 3 defined as input 5
INDUST4-1K      ; User status bit 4 defined as interrupt status bit 1
INDUST16-2I     ; User status bit 16 defined as system status bit 2
DEF bigmov      ; Define program bigmov
D20,20,1,3      ; Sets move distance on axes 1 and 2 to 20 units,
                ; axis 3 to 1 unit, and axis 4 to 3 units
GO1111          ; Initiate motion on all axes
END             ; End program definition
ONP bigmov      ; Set ON program to bigmov
ONUSxxx1        ; On user status bit #4 (interrupt status bit 1) GOSUB to
                ; the ONP program
ONCOND0100      ; Enable ONUS condition
```

ONVARA On Variable 1 Condition Gosub

Type	On Condition (Program Interrupt)
Syntax	<!>ONVARA<i,i>
Units	See below
Range	±999,999,999.99999999
Default	0,0
Response	ONVARA: *ONVARA0,0
See Also	ONCOND, ONP, ONVARB, VAR

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The On Variable 1 Condition Gosub (ONVARA) command specifies the low and high values which will cause a branch to the ON program (ONP). If the value of variable 1 is less than or equal to the first i, or greater than or equal to the second i, a GOSUB is performed. The subroutine or program that the GOSUB branches to is selected by the ON program (ONP) command.

The ONVARA command must be enabled using the ONCOND command before any branching will occur. Once a branch to the ONP program occurs, ONVARA command will not call the ONP program while the ONP program is executing, eliminating the possibility of recursive calls. After returning from the ONP program, variable 1 must be reset to a value within the low and high values before another branch to the ONP program, resulting from the value of variable 1, will be allowed.

Example:

```
DEF bigmov      ; Define program bigmov
D20,20,1,3      ; Sets move distance on axes 1 and 2 to 20 units,
                ; axis 3 to 1 unit, and axis 4 to 3 units
GO1111          ; Initiate motion on all axes
END             ; End program definition
ONP bigmov      ; Set ON program to bigmov
ONVARA0,12      ; On VAR1 <= 0, or VAR1 >= 12 GOSUB to ONP program
ONCOND0010      ; Enable ONVARA condition
```

ONVARB On Variable 2 Condition Gosub

Type	On Condition (Program Interrupt)
Syntax	<!>ONVARB<i,i>
Units	See below
Range	±999,999,999.99999999
Default	0,0
Response	ONVARB: *ONVARB0,0
See Also	ONCOND, ONP, ONVARA, VAR

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The ONVARB command specifies the low and high values which will cause a branch to the ON program (ONP). If the value of variable 2 is less than or equal to the first i, or greater than or equal to the second i, a GOSUB is performed. The subroutine or program that the GOSUB branches to is selected by the ON program (ONP) command.

The ONVARB command must be enabled using the ONCOND command before any branching will occur. Once a branch to the ONP program occurs, ONVARB command will not call the ONP program while the ONP program is executing, eliminating the possibility of recursive calls. After returning from the ONP program, variable 2 must be reset to a value within the low and high values before another branch to the ONP program, resulting from the value of variable 1, will be allowed.

Example:

```
DEF bigmov      ; Define program bigmov
D20,20,1,3      ; Sets move distance on axes 1 and 2 to 20 units,
                ; axis 3 to 1 unit, and axis 4 to 3 units
GO1111          ; Initiate motion on all axes
END             ; End program definition
ONP bigmov      ; Set ON program to bigmov
ONVARB0,12      ; On VAR2 <= 0, or VAR2 >= 12 GOSUB to ONP program
ONCOND0001      ; Enable ONVARB condition
```

[OR] Or

		Product	Rev
Type	Operator (Logical)	AT6n00	1.0
Syntax	See below	AT6n50	1.0
Units	n/a	610n	4.0
Range	n/a	615n	1.0
Default	n/a	620n	1.0
Response	n/a	625n	1.0
See Also	[AND], IF, [NOT], NWHILE, REPEAT, UNTIL, WAIT, WHILE	6270	1.0

Use the OR command as a logical operator in a program flow control command (IF, REPEAT, UNTIL, WHILE, NWHILE, WAIT). The OR command logically links two expressions. If either of the two expressions are true, and are linked with an OR command, then the whole statement is true. This fact is best illustrated by example.

If VAR1=1 and VAR2=1 then, even though variable 2 is not greater than 3, this is a true statement: IF(VAR1>0 OR VAR2>3). This statement would not be true: IF(VAR1<>1 OR VAR2=2).

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

TRUE OR TRUE = TRUE
TRUE OR FALSE = TRUE
FALSE OR TRUE = TRUE
FALSE OR FALSE = FALSE

Example:

```
VAR1=1                  ; Set variable 1 equal to 1
IF(VAR1=1 OR IN=b1XXX)  ; Compare variable 1 to 1, and check for input #1
                        ; to be active
WRITE"FIRST EXAMPLE"    ; If either condition is true, write out FIRST EXAMPLE
NIF                     ; End IF statement
```

OUT Output State

		Product	Rev
Type	Output	AT6n00	1.0
Syntax	<!>OUT<b><b><b>...<b><b><b>	AT6n50	1.0
Units	n/a	610n	4.0
Range	b = 0 (off), 1 (on) or X (don't change)	615n	1.0
Default	0	620n	1.0
Response	n/a	625n	1.0
See Also	OUTALL, OUTEN, OUTFNC, OUTLVL, OUTP, TOUT	6270	1.0

The Output State (OUT) command turns the programmable output bits on and off. A programmable output bit must be defined as such (OUTFNCi-A, which is the default definition) before this command will take affect.

The output bit pattern specified in the OUT command corresponds only to the outputs defined as programmable (OUTFNCi-A). For example, if only outputs 3 and 5 were defined as programmable, then turning on outputs 3 and 5 would require the command OUT11, not OUTxx1x1.

Output bit patterns vary by product. The output bit patterns for all 6000 products are provided on page 6 of this document.

If it is desired to set only one output value, instead of all outputs, the bit select (.) operator can be used, followed by the number of the specific output. For example, OUT.12-1 turns on output 12.

Example:

```
OUTFEN1         ; Enable output functions
OUTFNC1-1B      ; Define output #1 as axis 1 moving/not moving
OUTFNC2-2B      ; Define output #2 as axis 2 moving/not moving
OUTFNC3-A       ; Define output #3 as programmable
OUTFNC4-A       ; Define output #4 as programmable
OUTFNC5-F       ; Define output #5 as fault output
OUT10           ; Turn on first programmable output (output #3),
                ; turn off second programmable output (output #4)
OUT.2-1         ; Turn on the second programmable output (output #4)
```

[OUT] Output Status

Type	Assignment or Comparison
Syntax	See below
Units	n/a
Range	b = 0 (off), 1 (on) or X (don't change)
Default	0
Response	n/a
See Also	OUTALL, OUTEN, OUTFNC, OUTLVL, TOUT, VARB

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Output Status (OUT) command is used to assign the output states to a binary variable (VARB), or to make a comparison against a binary or hexadecimal value. To make a comparison against a binary value, the letter b (b or B) must be placed in front of the value. The binary value itself must only contain ones, zeros, or Xs (1, Ø, X, x). To make a comparison against a hexadecimal value, the letter h (h or H) must be placed in front of the value. The hexadecimal value itself must only contain the letters A through F, or the numbers Ø through 9.In servo products, the OUTEN command has no effect on auxiliary outputs (**OUT-A**, **OUT-B**, etc.) when they are configured as output-on-position outputs with the OUTFNCi-H command.

Syntax: VARBn=OUT where n is the binary variable number,
or [OUT] can be used in an expression such as IF(OUT=b11Ø1), or IF(OUT=h7F)

Output bit assignments vary by product. The output bit patterns for all 6000 products are provided on page 6 of this document. The outputs are numbered 1 to *n* (*n* depends on the product) from left to right.

The function of the outputs is established with the OUTFNC command (although the [OUT] command looks at all outputs regardless of their assigned function from the OUTFNC command). If it is desired to assign only one output value to a binary variable, instead of all outputs, the bit select (.) operator can be used, followed by the number of the specific output. For example, VARB1=OUT.12 assigns output 12 to binary variable 1.

Example:

```
VARB1=OUT           ; Output status assigned to binary variable 1
VARB2=OUT.12        ; Output bit 12 assigned to binary variable 2
VARB2               ; Response if bit 12 is set to 1:
                    ; *VARB2=XXXX_XXXX_XXX1_XXXX_XXXX_XXXX_XXXX_XXXX
IF(OUT=b111011X11)  ; If the output status contains 1's for outputs 1, 2, 3, 5,
                    ; 6, 8, and 9, and a 0 for output 4, do the IF statement
TREV                ; Transfer revision level
NIF                 ; End IF statement
IF(OUT=h7F00)       ; If the output status contains 1's for outputs 1, 2, 3, 5,
                    ; 6, 7, and 8, and 0's for every other output, do the IF
                    ; statement
TREV                ; Transfer revision level
NIF                 ; End IF statement
```

OUTALL Output State for Multiple Outputs

Type	Output
Syntax	<!>OUTALL<i>,<i>,<b>
Units	See below
Range	1st i = 1 to *n*; 2nd i = First i to *n*; b = 0 (off) or 1 (on) (*n* = total # of general-purpose outputs, varies by product and option)
Default	0
Response	n/a
See Also	OUT, OUTEN, OUTFNC, OUTLVL, TOUT

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Output State (OUTALL) command turns the programmable output bits on and off. A programmable output bit must be defined as such (OUTFNCi-A) before this command will take affect. **Note**: OUTFNCi-A is the default.

The outputs specified in the OUTALL command corresponds only to the outputs defined as programmable (OUTFNCi-A). For example, if only outputs 3 and 5 were defined as programmable, then stating the command OUTALL1,2,1 will turn on outputs 3 and 5.

Syntax: First i = Beginning of output set. Range: 1 to *n* *

Second i = Ending of output set. Range: first i to *n* *

b = State of the outputs (Ø = off, 1 = on)

* *n* represents the maximum number of programmable outputs. This number varies by product. The programmable output configurations for all 6000 products are provided on page 6 of this document.

Example:
```
OUTALL1,14,1    ; Turn on programmable outputs 1 through 14
```

OUTANA Analog Output, Auxiliary

Type	Output
Syntax	<!>OUTANA<r>
Units	r = Volts
Range	-10.000 to + 10.000
Default	0
Response	n/a
See Also	none

Product	Rev
AT6n00	n/a
AT6n50	1.0
610n	n/a
615n	n/a
620n	n/a
625n	n/a
6270	n/a

The OUTANA command controls the voltage setting from the ANA output on the AT6n50's AUX connector. The range is -10V to +10V with an accuracy of +/- 5%. The output is controlled with an 8-bit digital-to-analog converter.

Example:
```
OUTANA5.00      ; Set ANA output to +5.00 volts
OUTANA-3.00     ; Set ANA output to -3.00 volts
```

OUTEN Output Enable

Type	Output or Program Debug Tool
Syntax	<!>OUTEN<d><d><d>...<d><d><d>
Units	n/a
Range	d = 0 (off), 1 (on), E (enabled) or X (don't change)
Default	E
Response	OUTEN: *OUTENEEEE_EEEE_EEEE_EEEE_EEEE_EEEE
See Also	OUT, OUTFNC, OUTLVL, TSTAT

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Output Enable (OUTEN) command allows the user to disable any of the outputs from their configured function and set them on or off. This command is used for troubleshooting and initial start-up testing. It allows you to simulate output operations by bypassing the configured output function.

0 = Disable output function and turn output off
1 = Disable output function and turn output on
X = No change from previous state
E = Re-enable output function

In servo products, the OUTEN command has no effect on auxiliary outputs (**OUT-A**, **OUT-B**, etc.) when they are configured as output-on-position outputs with the OUTFNCi-H command.

Programmable output bit assignments vary by product. The output bit patterns for all 6000 products are provided on page 6 of this document. The outputs are numbered 1 to *n* (*n* depends on the product) from left to right.

Example:

```
; This allows the user to test if the fault output is working,
; without the inconvenience of trying to force a fault.
OUTFEN1         ; Enable output functions
OUTFNC1-1B      ; Define output #1 as axis 1 moving/not moving
OUTFNC2-2B      ; Define output #2 as axis 2 moving/not moving
OUTFNC3-A       ; Define output #3 as programmable
OUTFNC4-A       ; Define output #4 as programmable
OUTFNC5-F       ; Define output #5 as fault output
OUTENxxxx1      ; Disable programmed function of output #5 and turn on
```

OUTFEN Output Function Enable

Type	Output
Syntax	<!>OUTFEN<b>
Units	n/a
Range	b = 0 (disable), 1 (enable) or X (don't change)
Default	0
Response	OUTFEN: *OUTFEN0
See Also	OUTFNC, OUTP, POUT

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Output Function Enable (OUTFEN) command enables the use of the OUTFNC command, as well as the OUTPA, OUTPB, OUTPC, and OUTPD commands. If OUTFEN is disabled, the outputs can only be used as programmable outputs (OUTFNCi-A).

Example:

```
OUTFEN1         ; Enable output functions
```

OUTFNC Output Function

Type	Output
Syntax	<!>OUTFNC<i><-<a>c>
Units	i = output #, a = axis, c = function identifier (letter)
Range	i = up to 28 (depends on product), a = up to 4 (depends on product), c = A - H
Default	c = A (programmable output function)
Response	OUTFNC: *OUTFNC1-A PROGRAMMABLE OUTPUT - STATUS OFF ... (repeated once for each programmable output) *OUTFNC28-A PROGRAMMABLE OUTPUT - STATUS OFF OUTFNC1: *OUTFNC1-A PROGRAMMABLE OUTPUT - STATUS OFF
See Also	OUT, OUTEN, OUTFEN, OUTLVL, OUTP, OUTPLC, OUTTW, POUT, SMPER, TSTAT

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Output Function (OUTFNC) command defines the functions for each output. The factory setting for all the outputs is programmable output bits (OUTFNCi-A).

NOTE

The output functions must be enabled with the OUTFEN command. If the output functions are not enabled, only OUTFNCi-A will function correctly.

Programmable output bit assignments vary by product. The output bit patterns for all 6000 products are provided on page 6. The outputs are numbered 1 to *n* (*n* depends on the product) from left to right. The programmable outputs are scanned once per *system update* (steppers: 2 milliseconds; servos: depends on current INDAX and SSFR settings — see table in SSFR command description).

For the functions that are axis specific (B, D, and E), an optional axis specifier may be placed in front of the function. By placing the axis specifier in front of the function letter, the output will only go active when the specific axis specified has the corresponding condition. If an axis specifier is not specified, then if any of the axes have the corresponding condition, the output will go active. The output functions are as follows:

Identifier	Function Description
A	**Programmable Output**: Standard output (default function). Turn on or off with the OUT or OUTALL commands to affect external processes. To view the state of the outputs, use the TOUT command. To use the state of the outputs as a basis for conditional branching or looping statements (IF, REPEAT, WHILE, etc.), use the [OUT] command.
B	**Moving/Not Moving Axis**: Output activates when the axis is moving. As soon as the move is completed, the output will change to the opposite state. Servos: With the target zone mode enabled (STRGTE1), the output will not change state until the move completion criteria set with the STRGTD and STRGTV commands has been met. In this manner, the output functions as an *In Position* output.
C	**Program in Progress**: Output activates when a program is being executed. After the program is finished, the output's state is reversed.
D	**End-of-Travel Limit Encountered**: Output activates when a hard or soft end-of-travel limit has been encountered. When a limit is encountered, you will not be able to move the motor in that same direction until you clear the limit by changing direction (D) and issuing a GO command. (An alternative is to disable the limits with the LH0 command, but this is recommended only if the motor is not coupled to the load.)
E	**Stall Indicator** (Steppers Only): Output activates when a stall is detected. To detect a stall, you must first connect an encoder, enable the encoder step mode with the ENC1 command, enable the position maintenance function with the EPM1 command, and enable stall detection with the ESTALL1 command. For details refer to the *Closed-Loop Stepper Setup* section in the *6000 Series Programmer's Guide*. Not applicable to the OEM-AT6n00.
F	**Fault Indicator**: Output activates when either the user fault input or the drive fault input becomes active. The user fault input is a general-purpose input defined as a user fault input with the INFNCi-F command. Make sure the drive fault active level (DRFLVL) is appropriate for the drive you are using. The drive fault input is not available on the OEM-AT6n00.
G	**Position Error Exceeds Max. Limit** (Servos Only): Output activates when the maximum allowable position error, as defined with the SMPER command, is exceeded. The position error (TPER) is defined as the difference between the commanded position (TPC) and the actual position as measured by the feedback device. When the maximum position error is exceeded (usually due to instability or loss of position feedback from the feedback device), the controller shuts down the drive and sets error status bit #12 (reported by the TER command). If the SMPER command is set to zero (SMPER0), the position error will not be monitored; thus, the *Maximum Position Error Exceeded* function will not be usable.
H	**Output On Position** (Servos Only): Output activates when the specified axis is at a specified position. Applicable only to the auxiliary outputs (**OUT-A** through **OUT-D**). Output On Position function parameters are configured with the OUTPA, OUTPB, OUTPC, and OUTPD commands for axes 1 through 4, respectively. This function is not applicable to the OEM6250. Output On Position cannot be used with ANI feedback.

Example:

```
OUTFEN1          ; Enable output functions
OUTFNC1-3B       ; Define output #1 as axis 3 moving/not moving
OUTFNC2-D        ; Define output #2 to go active when any of the limits are
                 ; hit on any axis
```

OUTLVL Output Active Level

Type	Output
Syntax	<!>OUTLVL<b><b><b>...<b><b><b>
Units	n/a
Range	b = 0 (active low), 1 (active high) or X (don't change)
Default	0
Response	OUTLVL: *OUTLVL0000_0000_0000_0000_0000_0000
See Also	OUT, OUTEN, OUTFEN, OUTFNC, OUTP, OUTPLC, OUTTW, POUT

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Output Active Level (OUTLVL) command defines the active state of each programmable output. The default state is active low. Programmable output bit pattern varies by product — refer to page 6. Refer to the 6000 Series product *Installation Guide* for programmable output schematics.

When an output is defined to be active low, an OUT1 command will cause the output to be pulled to ground. When an output is defined to be active high, an OUT1 command will cause the output to source current from the power supply.

Example:

```
OUTLVL1x0  ; Config output 1 active high, output 2 unchanged, output 3 active low
```

OUTP Output on Position — Axis Specific

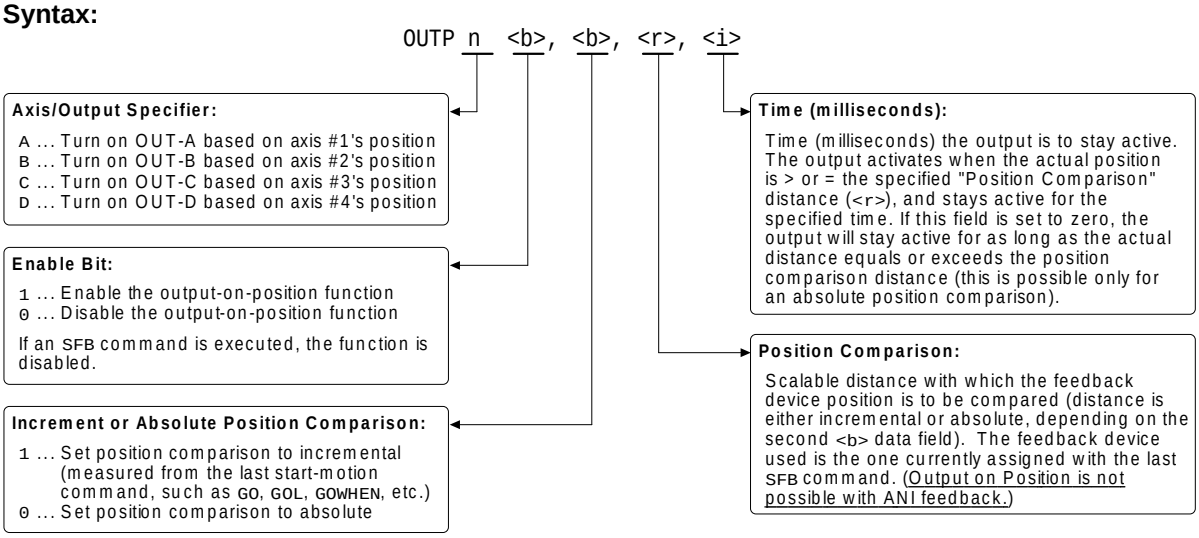
Type	Output
Syntax	<!>OUTPn<b>,<b>,<r>,<i>
Units	n = axis/output identifier letter b = enable bits; r = scalable distance; i = time (ms)
Range	n = A-D (A for OUT-A, axis 1, B for OUT-B, axis 2, etc.); b = 0 or 1; r = -999,999,999 - +999,999,999; i = 0 - 65535
Default	0,0,0,0
Response	OUTPA: *OUTPA0,0,+0,0
See Also	[OUT], OUT, OUTFEN, OUTFNC, PSET, SFB

Product	Rev
AT6n00	n/a
AT6n50	1.0
610n	n/a
615n	1.0
620n	n/a
625n	1.0
6270	1.0

Use the Output on Position (OUTPn) command to configure the respective auxiliary analog output (**OUT-A** through **OUT-D**) to activate based on the actual position of the respective axis. The position referenced is the position of the feedback device currently selected with the SFB command. If the SFB command is changed, the output-on-position function is disabled until a new OUTPn command re-enables the function.

To use the OUTPn command, you must first use the OUTFNCi-H command to configure the auxiliary output to function as an *output on position* output, and you must enable output functions with the OUTFEN1 command. (The "i" in the OUTFNCi-H command represents the number of the auxiliary output in the product's output bit pattern — see page 6 for output bit patterns for each product.) Refer to the programming example below.

Syntax:



NOTE

The output activates only during motion; therefore, issuing a PSET command to set the absolute position counter to activate the output on position will not turn on the output until the next motion occurs.

Example (for the AT6450):

```
OUTFEN1             ; Enable output functions
OUTFNC25-H          ; Set OUT-A (output #25) as an "output on position" output
OUTFNC26-H          ; Set OUT-B (output #26) as an "output on position" output
OUTPA1,0,+50000,50  ; Turn on OUT-A for 50 ms when the actual position of
                    ; axis #1 is > or = absolute position +50,000
OUTPB1,1,+30000,50  ; Turn on OUT-B for 50 ms when the actual position of
                    ; axis #1 is > or = incremental position 30,000 (since the
                    ; last GO)
```

OUTPLC Establish PLC Strobe Outputs

Type	Output
Syntax	<!>OUTPLC<i>,<i-i>,<i>,<i>
Units	See below
Range	See below
Default	1,0-0,0,0
Response	OUTPLC1: *OUTPLC1,0-0,0,0
See Also	INPLC, OUT, OUTEN, OUTFNC, OUTLVL, OUTTW, [TW]

Product	Rev
AT6n00	1/0
OEM-AT6n00	3.0
AT6n50	1/0
610n	4.0
615n	1/0
620n	1/0
625n	1/0
6270	1/0

The Establish PLC Strobe Outputs (OUTPLC) command with its corresponding INPLC command configure the applicable inputs and outputs to read data from a parallel I/O device such as a PLC (Programmable Logic Controller), or a passive thumbwheel module. The actual data transfer occurs with the TW command. Refer to the TW command for a description of the data transfer process.

The OUTPLC command has four fields (<i>,<i-i>,<i>,<i>):

Data Field	Description
Field 1: <i>	**Set #**: There are 4 possible OUTPLC sets (1-4). This field identifies which set to use.
Field 2: <i-i>	**Strobe Output #s**: Data reads with the TW command are strobed by the outputs selected in this field. The first number is the first output, and the second number is the last output. The outputs must be consecutive. The number of outputs should equal half the number of the maximum number of BCD digits required. If 6 digits are being read, then three outputs are needed as each output strobe selects two BCD digits.
Field 3: <i>	**TW Command Pending**: This field identifies an output that becomes active on a TW command and then turns off on completion of the TW command. This output can signal a device that a TW command is pending. A zero in this field will not activate any output.
Field 4: <i>	**Strobe Time**: This field identifies the length of time an output will stay active in order to read the BCD digits. The strobe time (in milliseconds) should be greater than the PLC scan time, if a PLC is being used, or set greater than the minimal debounce time if using thumbwheels. Range = 1 - 5000 milliseconds.

To disable a specific PLC set, enter OUTPLCn,0-0,0,0 where n is the PLC set (1-4).

Example:

```
INPLC2,1-8,9,10   ; Set INPLC set 2 as BCD digits on inputs 1 - 8, with input 9
                  ; as the sign bit, and input 10 as the data valid
OUTPLC2,1-4,5,50  ; Set OUTPLC set 2 as output strobes on outputs 1 - 4, with
                  ; output 5 as the command pending bit, and strobe time of 50
                  ; milliseconds
A(TW6)            ; Read data into axis 1 acceleration using INPLC set 2
                  ; and OUTPLC set 2 as the data configuration
```

OUTTW Establish Thumbwheel Strobe Outputs

Type	Output
Syntax	<!>OUTTW<i>,<i-i>,<i>,<i>
Units	See below
Range	See below
Default	1,0-0,0,0
Response	OUTTW1: *OUTTW1,0-0,0,0
See Also	INSTW, OUT, OUTEN, OUTFNC, OUTLVL, OUTPLC, [TW]

Product	Rev
AT6n00	1.0
OEM-AT6n00	3.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Establish Thumbwheel Strobe Outputs (OUTTW) command with its corresponding INSTW command configure the applicable inputs and outputs to read data from an active thumbwheel device such as Compumotor's TM8 Thumbwheel Module. The actual data transfer occurs with the TW command. Refer to the TW command for a description of the data transfer process.

The OUTTW command has four fields (<i>,<i-i>,<i>,<i>):

Data Field	Description
Field 1: <i>	**Set #**: There are 4 possible OUTTW sets (1-4). This field identifies which set to use.
Field 2: <i-i>	**Strobe Output #s**: Data reads with the TW command are strobed by the outputs selected in this field. The first number is the first output, and the second number is the last output. The outputs must be consecutive. The number of outputs should be compatible to the thumbwheel device (3 for the TM8 Module).
Field 3: <i>	**TM8 Enable Output**: This field identifies an output that becomes active on a TW command and then turns off on completion of the TW command. This output can enable a TM8 module to respond, thus allowing multiple TM8s to be wired to the inputs and outputs. A zero in this field will not activate any output.
Field 4: <i>	**Strobe Time**: This field identifies the length of time an output will stay active to read the BCD digits. The strobe time (in milliseconds) should be set to a minimal debounce time. Range = 1 - 5000 milliseconds.

Example:

```
INSTW2,1-4,5      ; Set INSTW set 2 as BCD digits on inputs 1 - 4, with input 5
                  ; as the sign bit
OUTTW2,1-3,4,50   ; Set OUTTW set 2 as output strobes on outputs 1 - 3, with
                  ; output 4 as the output enable bit, and strobe time of 50
                  ; milliseconds
A(TW2)            ; Read data into axis 1 acceleration using INSTW set 2 and
                  ; OUTTW set 2 as the data configuration
```

PA — Path Acceleration

Type	Path Contouring or Motion (Linear Interpolated)
Syntax	<!>PA<r>
Units	r = units/sec/sec
Range	0.00025 - 24,999,999 (depending on the scaling factor)
Default	10.0000
Response	PA: *PA10.0000
See Also	GOL, PAA, PAD, PADA, PSCLA, SCALE

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	n/a
615n	n/a
620n	1.0
625n	1.0
6270	1.0

The Path Acceleration (PA) command specifies the path acceleration to be used with linearly interpolated moves (GOL), and all contouring moves (PLIN, PARCM, PARCOM, PARCOP, PARCP). For both the linear interpolated and the contouring moves, the path acceleration refers to the acceleration experienced by the load as motion gains speed along the path. For linearly interpolated moves, the acceleration of each individual axis is dependent on the distance it contributes to the total path traveled by the load. In contouring paths, the acceleration of each individual axis is dependent on the direction of travel in the X-Y plane. **NOTE:** *The PA value can be altered between path segments, but not within a path segment.*

Contouring and linear interpolation are discussed in detail in the Custom Profiling chapter of the 6000 Series Programmer's Guide.

Steppers: The entered value is always in reference to motor steps, not encoder steps, regardless of the ENC command setting. If scaling is not enabled (SCALEØ), the acceleration value is entered in motor revs/sec/sec ; this value is internally multiplied by the drive resolution (DRES) value to obtain an acceleration value in motor steps/sec/sec for the motor trajectory calculations. If scaling is enabled (SCALE1), the entered acceleration value is internally multiplied by the path acceleration scaling factor (PSCLA) to convert user units/sec/sec to motor steps/sec/sec.

Servos: If scaling is not enabled (SCALEØ), the acceleration value is entered in encoder or resolver revs/sec/sec, LDT inches/sec/sec, or ANI volts/sec/sec; encoder or resolver and LDT values are internally multiplied by the encoder resolution (ERES) value or the LDT resolution (LDTRES) value to obtain an acceleration value in steps/sec/sec for the motion trajectory calculations. If scaling is enabled (SCALE1), the entered acceleration value is internally multiplied by the acceleration scaling factor (SCLA) to convert user units/sec/sec to encoder, resolver, LDT, or ANI steps/sec/sec.

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

If the path deceleration (PAD) command has not been entered, the path acceleration (PA) command will set the path deceleration rate. Once the path deceleration (PAD) command has been entered, the path acceleration (PA) command no longer affects path deceleration.

Example:

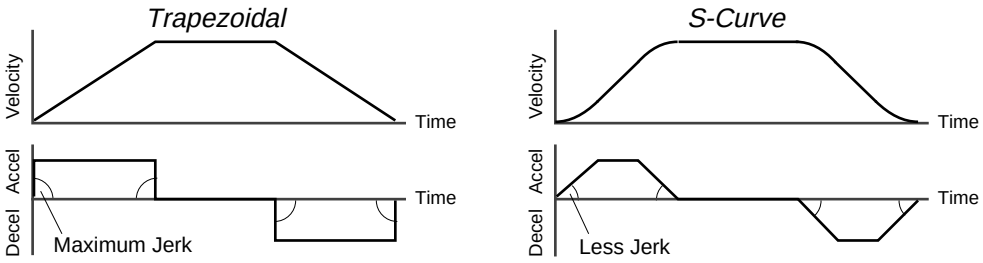
```
PSCLA25000      ; Set path acceleration scale factor to 25000 steps/unit
PSCLD25000      ; Set path distance scale factor to 25000 steps/unit
PSCLV25000      ; Set path velocity scale factor to 25000 steps/unit
SCALE1          ; Enable scaling factor
PV5             ; Set path velocity to 5 units/sec
PA50            ; Set path acceleration to 50 units/sec/sec
PAD100          ; Set path deceleration to 100 units/sec/sec
DEF prog1       ; Begin definition of path named prog1
PAXES1,2        ; Set axes 1 and 2 as the X and Y contouring axes
PAB0            ; Set to incremental coordinates
PLIN1,1         ; Specify X-Y endpoint position to create a 45 degree
                ; angle line segment
END             ; End definition of path prog1
PCOMP prog1     ; Compile path prog1
PRUN prog1      ; Execute path prog1
```

PAA — Path Average Acceleration

Type	Motion (S-Curve) or Motion (Linear Interpolated)
Syntax	<!><@><a>PAA<r>,<r>,<r>,<r>
Units	r = units/sec/sec
Range	0.00025 - 24999999 (depending on the scaling factor)
Default	10.00 (trapezoidal profiling is default, where PAA tracks PA)
Response	PAA: *PAA10.0000,10.0000,10.0000,10.0000 1PAA: *1PAA10.0000
See Also	DRES, PA, PAD, PADA, PSCLA, SCALE

Product	Rev
AT6n00	n/a
AT6n50	1.0
610n	n/a
615n	n/a
620n	n/a
625n	1.0
6270	1.0

The Path Average Acceleration (PAA) command allows you to specify the average acceleration for an S-curve path profile. S-curve profiling provides smoother motion control by reducing the rate of change in acceleration and deceleration; this accel/decel rate of change is known as *jerk*. S-curve profiling improves position tracking performance in linear interpolation applications. S-curve profiling is not available for contouring applications.



The values for the maximum path accel (PA) and average path accel (PAA) commands determine the characteristics of the S-curve. To smooth the acceleration ramp, you must enter a PAA command value that satisfies this equation: 1/2 PA ≤ PAA < PA. The following conditions are possible:

Acceleration Setting	Profiling Condition
PAA > 1/2 PA, but PAA < PA	S-curve profile with a variable period of constant acceleration
PAA = 1/2 PA	*Pure S-curve* (no period of constant acceleration—smoothest motion)
PAA = PA	Trapezoidal profile (but can be changed to an S-curve by specifying a new PAA value less than PA)
PAA < 1/2 PA; or PAA > PA	When you issue a GOL command, the move will not be executed and an error message, *INVALID CONDITIONS FOR S_CURVE ACCELERATION–FIELD *n*, will be displayed.
PAA = zero	S-curve profiling is disabled. Trapezoidal profiling is enabled. PAA tracks PA, & PADA tracks PAD. (*Track* means the command's value will match the other command's value and will continue to match whatever the other command's value is set to.)
No PAA value ever entered	Profile will default to trapezoidal. PAA tracks PA.

While programming S-curves, if you never change the maximum or average path deceleration (PAD or PADA) commands, PADA will track PAA. However, once you change PAD, PADA will no longer track changes in PAA.

NOTE

Once you enter a PAA value that is ≠ zero or ≠ PA, S-curve profiling is enabled **only for interpolated moves** (e.g., not for homing, which requires the HOMADA and/or HOMAA commands). All subsequent interpolated moves for that axis must comply with this equation: 1/2 PA ≤ PAA < PA.

Increasing the PAA value above the pure S-curve level (PAA > 1/2 PA), the time required to reach the target velocity and the target distance is decreased. However, increasing PAA also increases jerk.

The calculation for determining S-curve average accel and decel move times is as follows:

(A_{avg} = average accel or decel value):

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

NOTE: Path acceleration scaling (PSCLA) affects PAA the same as it does for PA.

*** For a more in-depth discussion on S-curve profiling, refer to the Custom Profiling chapter in the *6000 Series Programmer's Guide*.

Example:

```
PSCLA25000      ; Set path acceleration scale factor to 25000 steps/unit
SCLD1000        ; Set distance scale factor to 1000 steps/unit, all axes
PSCLV25000      ; Set path velocity scale factor to 25000 steps/unit
SCALE1          ; Enable scaling factor
PV5             ; Set path velocity to 5 units/sec
PA50            ; Set path acceleration to 50 units/sec/sec
PAA40           ; Set path s-curve (average) acceleration to 40 units/sec/sec
PAD100          ; Set path deceleration to 100 units/sec/sec
PADA70          ; Set path s-curve (average) deceleration to 70 units/sec/sec
DEF prog1       ; Begin definition of path named prog1
D10,5,2,11      ; Set distance values, axes 1-4
GOL1111         ; Initiate linear interpolation motion
END             ; End definition of path prog1
```

PAB — Path Absolute

Type	Path Contouring
Syntax	<!>PAB<b>
Units	n/a
Range	b = 0 (incremental) or 1 (absolute)
Default	0
Response	No response - Must be defining a path (DEF)
See Also	PL, PLC, PSCLD, PWC, SCALE

Product	Rev
AT6n00	1.0
AT6n50	4.1
610n	n/a
615n	n/a
620n	1.0
625n	4.1
6270	4.1

The Path Absolute (PAB) command is used to indicate whether the subsequent segment endpoints are specified in either incremental (Ø) or absolute (1) coordinates. Segment endpoint position specifications may be either absolute with respect to the user-defined coordinate system, or incremental, relative to the start of each individual segment. At any point along a path definition, coordinates may be switched from incremental to absolute.

The absolute coordinate system may be either the *work* coordinate system or the *local* coordinate system (see PL).

PAD — Path Deceleration

		Product	Rev
Type	Path Contouring or Motion (Linear Interpolated)	AT6n00	1.0
Syntax	<!>PAD<r>	AT6n50	1.0
Units	r = units/sec/sec	610n	n/a
Range	0.00025 - 24,999,999 (depending on the scaling factor)	615n	n/a
Default	10.0000 (PAD tracks PA)	620n	1.0
Response	PAD: *PAD10.0000	625n	1.0
See Also	GOL, PA, PAA, PADA, PSCLA, SCALE	6270	1.0

The Path Deceleration (PAD) command specifies the path deceleration to be used with linearly interpolated moves (GOL), and all contouring moves (PLIN, PARCM, PARCOM, PARCOP, PARCP). For both the linear interpolated and the contouring moves, the path deceleration refers to the deceleration experienced by the load as motion slows along the path. For linearly interpolated moves, the deceleration of each individual axis is dependent on the distance it contributes to the total path traveled by the load. In contouring paths, the deceleration of each individual axis is dependent on the direction of travel in the X-Y plane.

Steppers: The entered value is always in reference to motor steps, not encoder steps, regardless of the ENC command setting. If scaling is not enabled (SCALEØ), the deceleration value is entered in motor revs/sec/sec; this value is internally multiplied by the drive resolution (DRES) value to obtain an deceleration value in motor steps/sec/sec for the motor trajectory calculations. If scaling is enabled (SCALE1), the deceleration value is internally multiplied by the path acceleration scaling factor (PSCLA) to convert user units/sec/sec to motor steps/sec/sec.

Servos: If scaling is not enabled (SCALEØ), the deceleration value is entered in encoder or resolver revs/sec/sec, LDT inches/sec/sec, or ANI volts/sec/sec; encoder or resolver and LDT values are internally multiplied by the encoder resolution (ERES) value or the LDT resolution (LDTRES) value to obtain an acceleration value in steps/sec/sec for the motion trajectory calculations. If scaling is enabled (SCALE1), the entered deceleration value is internally multiplied by the deceleration scaling factor (SCLA) to convert user units/sec/sec to encoder, resolver, LDT, or ANI steps/sec/sec.

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

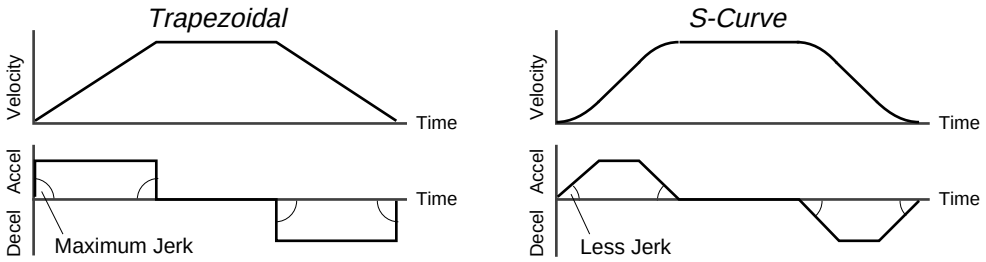
If the path deceleration (PAD) command has not been entered, the path acceleration (PA) command will set the path deceleration rate. Once the path deceleration (PAD) command has been entered, the path acceleration (PA) command no longer affects path deceleration. If PAD is set to zero (PADØ), then the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

Example: Refer to the path acceleration (PA) command example.

PADA — Path Average Deceleration

		Product	Rev
Type	Motion (S-Curve) or Motion (Linear Interpolated)	AT6n00	n/a
Syntax	<!><@><a>PADA<r>,<r>,<r>,<r>	AT6n50	1.0
Units	r = units/sec/sec	610n	n/a
Range	0.00025 - 24999999 (depending on the scaling factor)	615n	n/a
Default	10.00 (PADA tracks PAA)	620n	n/a
Response	PADA: *PADA10.0000,10.0000,10.0000,10.0000	625n	1.0
	1PADA: *1PADA10.0000	6270	1.0
See Also	DRES, PA, PAA, PAD, PSCLA, SCALE		

Use the Path Average Deceleration (PADA) command to specify the average deceleration for an S-curve path profile. S-curve profiling provides smoother motion control by reducing the rate of change in acceleration and deceleration; this accel/decel rate of change is known as *jerk*. S-curve profiling can improve position tracking performance in linear interpolation applications. S-curve profiling is not available for contouring applications.



The values for the path maximum decel (PAD) and path average decel (PADA) commands determine the characteristics of the S-curve. To smooth the deceleration ramp, you must enter an PADA command value that satisfies this equation: 1/2 PAD ≤ PADA < PAD. The following conditions are possible:

Deceleration Setting	Profiling Condition
PADA > 1/2 PAD, but PADA < PAD	S-curve profile with a variable period of constant deceleration
PADA = 1/2 PAD	*Pure S-curve* (no period of constant deceleration—smoothest motion)
PADA = PAD	Trapezoidal profile (but can be changed to S-curve by specifying a new PADA value < PAD)
PADA < 1/2 PAD; or PADA > PAD	When you issue a GOL command, the move will not be executed and an error message, *INVALID CONDITIONS FOR S_CURVE ACCELERATION–FIELD *n*, will be displayed.
PADA = zero	Upon entering the PADAØ command, an error message, *INVALID DATA–FIELD *n* will be displayed.
S-curve profiling with PAA, and no PADA or PAD ever entered	PADA will always match the PAA command value (identical S-curve accel and decel profiles). When you change PAD, PADA will no longer match changes in PAA.

NOTE

Once you enter a PADA value that is ≠ zero or ≠ PAD, S-curve profiling is enabled **only for interpolated move decelerations** (e.g., not for homing decelerations, which require the HOMADA command). All subsequent interpolated moves for that axis must comply with this equation: 1/2 PAD ≤ PADA < PAD.

Increasing the PADA value above the pure S-curve level (PADA > 1/2 PAD), the time required to reach the target velocity and the target distance is decreased. However, increasing PADA also increases jerk.

The calculation for determining S-curve average accel and decel move times is as follows:

(A_{avg} = average accel or decel value):

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

NOTE: Path acceleration scaling (PSCLA) affects PADA the same as it does for PAD.

*** For a more in-depth discussion on S-curve profiling, refer to the Custom Profiling chapter in the *6000 Series Programmer's Guide*.

Example: Refer to the path average acceleration (PAA) command example.

[PANI] Position of ANI Inputs *(-ANI Option Only)*

		Product	Rev
Type	Assignment or comparison	AT6n00	n/a
Syntax	See below	AT6n50-ANI	3.3
Units	n/a	610n	n/a
Range	n/a	615n-ANI	3.3
Default	n/a	620n	n/a
Response	n/a	625n-ANI	3.3
See Also	[ANI], ANIPOL, [FB], [CA], CMDDIR, PSET, SCALE, SCLD, SFB, TANI, TPANI, TFB	6270-ANI	3.0

The PANI command is used to assign the ANI analog inputs position information to a variable, or to make a comparison against another value. The PANI value represents the ANI input position after the affects of distance scaling (SCLD), offset (PSET), polarity (ANIPOL), and commanded direction polarity (CMDDIR).

The TPANI and PANI commands are designed for applications in which ANI input is scaled and/or used as position feedback. If you are using ANI input to monitor an analog signal, the TANI and ANI commands would be more appropriate (TANI and ANI values are measured in volts and are unaffected by scaling, offset, polarity, or command direction polarity).

The PANI value is represented in analog-to-digital converter (ADC) units if scaling is disabled (SCALEØ). The ADC has a 14-bit resolution, giving a range of +8191 to -8192 counts when using the full ±10V range of the ANI input (819 counts/volt). If scaling is enabled (SCALE1), an SCLD scale factor of 819 (the default value when ANI feedback is selected) allows units of volts to be used.

Syntax: VARn=aPANI where n is the variable number, and a is the analog input number (1-4, product dependent), or the PANI command can be used in an expression such as IF(1PANI=2.3). An analog input number specifier must precede the PANI command (e.g., 1PANI, 2PANI, etc.), or else it will default to input 1.

Example:

```
SCLD819          ; Set distance scaling to accommodate values in volts
                 ; (819 counts/volt)
SCALE1           ; Enable scaling
VAR4=2PANI       ; Position of ANI analog input #2 is assigned to variable 4
IF(1PANI<8.2)    ; If position of ANI input #1 is < 8.2 volts, do the
                 ; commands between the IF statement and the NIF statement.
TREV             ; Transfer revision level
NIF              ; End if statement
```

PARCM Radius Specified CCW Arc Segment

		Product	Rev
Type	Path Contouring	AT6n00	1.0
Syntax	<!>PARCM<r>,<r>,<r>	AT6n50	4.1
Units	r = units (see below)	610n	n/a
Range	0.00000 - ±999,999,999	615n	n/a
Default	n/a	620n	1.0
Response	No response - Must be defining a path (DEF)	625n	4.1
See Also	PARCP, PARCOM, PARCOP, PRTOL, PSCLD, SCALE	6270	4.1

The Radius Specified CCW Arc Segment (PARCM) command is used to specify the endpoints and the radius of a counter-clockwise arc segment. The placement, length, radius of curvature, and orientation of the arc are completely specified by the endpoint and radius specifications of the arc segment and the endpoint of the previous segment (current position). The direction of rotation in the X-Y plane will be counter-clockwise.

A complete circle cannot be specified with a PARCM command, because the center is arbitrary. ***Use the PARCOM command for circles.***

Command Syntax: PARCM<Xend>,<Yend>,<Radius>

Segment endpoint position specifications may be either absolute (PAB1) with respect to user defined segment start coordinates, or incremental (PABØ), relative to the start of each individual segment. The first two numbers following the PARCM command specify the X endpoint and the Y endpoint, respectively.

Radius specifications are signed values. A positive radius specifies an arc which is 180 degrees or less. A negative radius specifies an arc which is 180 degrees or more. The last number of the PARCM command specifies the radius.

Steppers only: All three position values are expressed in terms of motor steps, regardless of the current ENC command setting.

Scaling: If scaling (SCALE) is enabled, the PARCM command values entered are internally multiplied by the path distance scaling factor (PSCLD) to convert user units to motor steps. The distance values may be truncated if the values entered exceed the distance resolution at the given scaling factor. For further discussion on path distance scaling, refer to the PSCLD command description.

Example

```
PSCLA25000      ; Set path acceleration scale factor to 25000 steps/unit/unit
PSCLD25000      ; Set path distance scale factor to 25000 steps/unit
PSCLV25000      ; Set path velocity scale factor to 25000 steps/unit
SCALE1          ; Enable scaling factor
PV5             ; Set path velocity to 5 units/sec
PA50            ; Set path acceleration to 50 units/sec/sec
PAD100          ; Set path deceleration to 100 units/sec/sec
PSET0,0         ; Set absolute position to 0,0
DEF prog1       ; Begin definition of path named prog1
PAXES1,2        ; Set axes 1 and 2 as the X and Y contouring axes
PAB0            ; Set to incremental coordinates
POUT1001        ; Output pattern during first arc
PARCM5,5,5      ; Specify incremental X-Y endpoint position and radius arc
                ; <180 degrees for 1/4 circle counter-clockwise arc
POUT1100        ; Output pattern during second arc
PARCP5,-5,-5    ; Specify incremental X-Y endpoint position and radius arc
                ; >180 degrees for 3/4 circle clockwise arc
END             ; End definition of path prog1
PCOMP prog1     ; Compile path prog1
PRUN prog1      ; Execute path prog1
OUT0000         ; Turn off the first four programmable outputs
```

PARCOM Origin Specified CCW Arc Segment

Type	Path Contouring
Syntax	<!>PARCOM<r>,<r>,<r>,<r>
Units	r = units
Range	0.00000 - ±999,999,999
Default	n/a
Response	No response - Must be defining a path (DEF)
See Also	PARCOP, PARCM, PARCP, PRTOL, PSCLD, SCALE

Product	Rev
AT6n00	1.0
AT6n50	4.1
610n	n/a
615n	n/a
620n	1.0
625n	4.1
6270	4.1

The Origin Specified CCW Arc Segment (PARCOM) command is used to specify the coordinates necessary to create a counter-clockwise arc segment. The placement, length, radius of curvature, and orientation of the arc are completely specified by the endpoint and center specifications of the arc segment and the endpoint of the previous segment (current position). The direction of rotation in the X-Y plane will be counter-clockwise.

Command Syntax: PARCOM<Xend>,<Yend>,<Xcenter>,<Ycenter>

Segment endpoint position specifications may be either absolute (PAB1) with respect to user defined segment start coordinates, or incremental (PABØ), relative to the start of each individual segment. The first two numbers following the PARCOM command specify the X endpoint and the Y endpoint, respectively.

Center position specifications are always incremental, relative to the start of the arc segment. The last two numbers following the PARCOM command specify the X center point and Y center point coordinates, respectively.

Steppers only: All four position values are expressed in terms of motor steps, regardless of the current ENC command setting.

Scaling: If scaling (SCALE) is enabled, the PARCOM command values entered are internally multiplied by the path distance scaling factor (PSCLD) to convert user units to motor steps. The distance values may be truncated if the values entered exceed the distance resolution at the given scaling factor. For further discussion on path distance scaling, refer to the PSCLD command description.

Example:

```
PSCLA25000      ; Set path acceleration scale factor to 25000 steps/unit/unit
PSCLD25000      ; Set path distance scale factor to 25000 steps/unit
PSCLV25000      ; Set path velocity scale factor to 25000 steps/unit
SCALE1          ; Enable scaling factor
PV5             ; Set path velocity to 5 units/sec
PA50            ; Set path acceleration to 50 units/sec/sec
PAD100          ; Set path deceleration to 100 units/sec/sec
PSET0,0         ; Set absolute position to 0,0
DEF prog1       ; Begin definition of path named prog1
PAXES1,2        ; Set axes 1 and 2 as the X and Y contouring axes
PAB0            ; Set to incremental coordinates
POUT1001        ; Output pattern during first arc
PARCOM5,5,0,5   ; Specify incremental X-Y endpoint position and X-Y center
                ; position for quarter circle counter-clockwise arc
POUT1100        ; Output pattern during second arc
PARCOP0,0,5,0   ; Specify incremental X-Y endpoint position and X-Y center
                ; position for full circle clockwise arc
END             ; End definition of path prog1
PCOMP prog1     ; Compile path prog1
PRUN prog1      ; Execute path prog1
OUT0000         ; Turn off the first four programmable outputs
```

PARCOP Origin Specified CW Arc Segment

Type	Path Contouring
Syntax	<!>PARCOP<r>,<r>,<r>,<r>
Units	r = units
Range	0.00000 - ±999,999,999
Default	n/a
Response	No response - Must be defining a path (DEF)
See Also	PARCOM, PARCM, PARCP, PRTOL, PSCLD, SCALE

Product	Rev
AT6n00	1.0
AT6n50	4.1
610n	n/a
615n	n/a
620n	1.0
625n	4.1
6270	4.1

The Origin Specified CW Arc Segment (PARCOP) command is used to specify the coordinates necessary to create a clockwise arc segment. The placement, length, radius of curvature, and orientation of the arc are completely specified by the endpoint and center specifications of the arc segment and the endpoint of the previous segment (current position). The direction of rotation in the X-Y plane will be clockwise.

Command Syntax: PARCOP<Xend>,<Yend>,<Xcenter>,<Ycenter>

Segment endpoint position specifications may be either absolute (PAB1) with respect to user defined segment start coordinates, or incremental (PABØ), relative to the start of each individual segment. The first two numbers following the PARCOP command specify the X endpoint and the Y endpoint, respectively.

Center position specifications are always incremental, relative to the start of the arc segment. The last two numbers following the PARCOP command specify the X center point and Y center point coordinates, respectively.

Steppers only: All four position values are expressed in terms of motor steps, regardless of the current ENC command setting.

Scaling: If scaling (SCALE) is enabled, the PARCOP command values entered are internally multiplied by the path distance scaling factor (PSCLD) to convert user units to motor steps. The distance values may be truncated if the values entered exceed the distance resolution at the given scaling factor. For further discussion on path distance scaling, refer to the PSCLD command description.

Example: Refer to the PARCOM command example.

PARCP Radius Specified CW Arc Segment

		Product	Rev
Type	Path Contouring	AT6n00	1.0
Syntax	<!>PARCP<r>,<r>,<r>	AT6n50	4.1
Units	r = units	610n	n/a
Range	0.00000 - ±999,999,999	615n	n/a
Default	n/a	620n	1.0
Response	No response - Must be defining a path (DEF)	625n	4.1
See Also	PARCM, PARCOM, PARCOP, PRTOL, PSCLD, SCALE	6270	4.1

The Radius Specified CW Arc Segment (PARCP) command is used to specify the endpoints and the radius of a clockwise arc segment. The placement, length, radius of curvature, and orientation of the arc are completely specified by the endpoint and radius specifications of the arc segment and the endpoint of the previous segment (current position). The direction of rotation in the X-Y plane will be clockwise.

A complete circle cannot be specified with a PARCP command, because the center is arbitrary. ***Use the PARCOP command for circles.***

Command Syntax: PARCP<Xend>,<Yend>,<Radius>

Segment endpoint position specifications may be either absolute (PAB1)with respect to user defined segment start coordinates, or incremental (PABØ), relative to the start of each individual segment. The first two numbers following the PARCP command specify the X endpoint and the Y endpoint, respectively.

Radius specifications are signed values. A positive radius specifies an arc which is 180 degrees or less. A negative radius specifies an arc which is 180 degrees or more. The last number of the PARCP command specifies the radius.

Steppers only: All three position values are expressed in terms of motor steps, regardless of the current ENC command setting.

Scaling: If scaling (SCALE) is enabled, the PARCP command values entered are internally multiplied by the path distance scaling factor (PSCLD) to convert user units to motor steps. The distance values may be truncated if the values entered exceed the distance resolution at the given scaling factor. For further discussion on path distance scaling, refer to the PSCLD command description.

Example: Refer to the PARCM command example.

PAXES Set Contouring Axes

		Product	Rev
Type	Path Contouring	AT6n00	1.0
Syntax	<!>PAXES<i>,<i>,<i>,<i>	AT6n50	4.1
Units	See below	610n	n/a
Range	i = 1 - 4 (product dependent)	615n	n/a
Default	1,2,0,0	620n	1.0
Response	No response - Must be defining a path (DEF)	625n	4.1
See Also	DEF, END, PCOMP, PPRO, PRUN	6270	4.1

The Set Contouring Axes (PAXES) command defines the axes to be used in the current path definition. The four numbers following the comma specify the X, Y, Tangent, and Proportional axes, respectively. The X and Y axes must be specified, but the Tangent and Proportional axes are optional.

If no axis number is specified for the Tangent or Proportional axes, it signifies that the Tangent or Proportional axes are not included in that path definition. The axis specification for the entire path is done with this command. The PAXES command should be given prior to any contour segments.

NOTE: For 6000 Series products that control only 2 axes of motion, the Tangent and Proportional axes are not available.

Command Syntax: PAXES<Xaxis>,<Yaxis>,<Tangent>,<Proportional>

Example:

```
DEF prog1       ; Begin definition of path named prog1
PAXES1,2,3,4    ; Set axes 1,2,3,4 as the X, Y, Tangent, and
                ; Proportional axes respectively
PPRO2.25        ; Proportional axis path ratio = 2.25
                ; ************************************
                ; *  put                             *
                ; *  Multiple Segment Definitions    *
                ; *  here                            *
                ; ************************************
END             ; End definition of path prog1
PCOMP prog1     ; Compile path prog1
PRUN prog1      ; Execute path prog1
```

[PC] Position Commanded

		Product	Rev
Type	Assignment or Comparison		
Syntax	See below	AT6n00	n/a
Units	n/a	AT6n50	1.0
Range	n/a	610n	n/a
Default	n/a	615n	1.0
Response	n/a	620n	n/a
See Also	ERES, GOWHEN, [PCC], [PE], [PER], PSET, SCALE, SCLD, SMPER, TAS, TFB, TPC, TPCC, TPE, TPER	625n	1.0
		6270	1.0

The Position Commanded (PC) command is used to assign the current *commanded position* of each axis to a variable, or to make a comparison against another value. The value assigned to the variable or the value against which the comparison is made is measured in encoder steps and is scaled by the distance scaling factor (SCLD), if scaling is enabled with the SCALE1 command.

The commanded position is determined by the controller's move profile routine. The position profile is the *command* to the servo system that the motor must follow. The *actual position* is the position read by the feedback device (see TFB). The commanded position and the actual position are used in the control algorithm to determine the control signal. The position error is derived from the difference between the commanded position and the actual position (see TPER or PER).

Syntax: VARn=aPC where n is the variable number, and a is the axis, or [PC] can be used in an expression such as IF(1PC>5Ø). The PC command must be used with an axis specifier or it will default to axis 1 (e.g., 1PC, 2PC, etc.).

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

Example:

```
VAR1=1PC        ; Commanded position for axis 1 is assigned to variable 1
IF(2PC<50)      ; If the commanded position for axis 2 is <50, do the IF statement
VAR2=2PC+500    ; Commanded position for axis 2 plus 500 is assigned to variable 2
NIF             ; End IF statement
```

[PCA] Position of Captured ANI Input

Type	Assignment or Comparison
Syntax	See below
Units	n/a
Range	-10 to +10
Default	n/a
Response	n/a
See Also	[ANI], ANIPOL, [CA], CMDDIR, INFNC, [PANI], [PCA], PSET, SCLD, SFB, [SS], SSFR, TCA, TPANI, TPCA, TSS

Product	Rev
AT6n00	n/a
AT6n50-ANI	1.0
610n	n/a
615n-ANI	1.0
620n	n/a
625n-ANI	3.0
6270-ANI	1.0

The Position of Captured ANI (PCA) command is used to assign one of the captured ANI analog input register values (captured when trigger A, B, C or D is activated) to a variable, or to make a comparison against another value. Once the captured ANI register value is assigned to a variable, or a comparison is made, the respective position capture status (reported with bits 25 - 28 in the TSS or SS commands) is cleared, but the position information remains in the register until it is overwritten by a subsequent position capture from the trigger input.

The ANI value is referenced in counts and can be affected by:

- Distance scale (SCLD), if scaling is enabled (SCALE1)
- Position offset (PSET)
- ANI input polarity (ANIPOL)
- Commanded direction polarity (CMDDIR)

Syntax: VARn=aPCAc where n is the variable number, a is the axis, and c designates trigger A, B, C or D; or [PCA] can be used in an expression such as IF(1PCAB>5000).
The PCA command must be used with an analog input specifier or it will default to analog input 1 (e.g., 1PCAA, 2PCAB, etc.).

The ANI input value can be captured only by a trigger input signal (trigger A, B, C or D). The appropriate trigger inputs must be defined as *trigger interrupt* inputs with the INFNCi-H command, where i is the input bit number of the trigger input (input bit assignments vary by product—see page 6 for your product's bit assignment configuration). Once defined, an active signal on the specified trigger input will capture the ANI values on all axes. The ANI information is stored in registers and is available at the next system update through the use of the PCA and TPCA commands.

POSITION CAPTURE ACCURACY

If ANI feedback is selected with the SFB command, the captured ANI value is interpolated from the last sampled ANI input value and rate of change of the ANI input value, and the time elapsed since the last sample. *The position sample rate is determined by the SSFR and INDAX commands (system update rate). The accuracy of the position capture is ±50μs x velocity.*

If ANI feedback is NOT selected with the SFB command, the last sampled ANI value is simply stored as the captured ANI value. *The accuracy is one system update period (determined by SSFR and INDAX).*

If you issue a PSET (establish absolute position reference) command, any previously captured ANI input values will be offset by the value specified in the PSET command.

Example (for the 6250):

```
INFNC25-H       ; Assign trigger input A (TRG-A) as a trigger interrupt input
INFNC26-H       ; Assign trigger input B (TRG-B) as a trigger interrupt input
VAR1=1PCAA      ; Assign captured ANI value on analog input 1
                ; (captured when the TRG-A input became active) to variable 1
IF(2PCAB<40)    ; If the captured ANI value on analog input 2
                ; (captured when the TRG-B input became active) is less
                ; than 40, do the IF statement
VAR2=1PCAA+10   ; Add 10 to the captured ANI value on analog input 1
                ; (captured when the TRG-A input became active) and assign
                ; the sum to variable #2
NIF             ; End IF statement
```

[PCC] Captured Commanded Position

		Product	Rev
Type	Assignment or Comparison	AT6n00	n/a
Syntax	See below	AT6n50	1.0
Units	n/a	610n	n/a
Range	n/a	615n	1.0
Default	n/a	620n	n/a
Response	n/a	625n	3.0
See Also	CMDDIR, INFNC, [PC], PSET, SCALE, SCLD, SFB, [SS], SSFR, TPC, TPCC, TSS	6270	1.0

The Captured Commanded Position (PCC) command is used to assign one of the captured commanded position register values (captured when trigger A, B, C or D is activated) to a variable, or to make a comparison against another value. Once the captured commanded position register value is assigned to a variable, or a comparison is made, the respective position capture status (reported with bits 25 - 28 in the TSS or SS commands) is cleared, but the position information remains in the register until it is overwritten by a subsequent position capture from the trigger input.

If scaling is enabled (SCALE1), the commanded position is scaled by the distance scaling factor (SCLD). If scaling is not enabled (SCALEØ), the value assigned will be actual commanded counts.

Syntax: VARn=aPCCc where n is the variable number, a is the axis, and c designates trigger A, B, C or D; or [PCC] can be used in an expression such as IF(1PCCB>2345Ø). The PCC command must be used with an axis specifier or it will default to axis 1 (e.g., 1PCCA, 2PCCB, etc.).

The commanded position can be captured only by a trigger input signal (trigger A, B, C or D). The appropriate trigger inputs must be defined as *trigger interrupt* inputs with the INFNCi-H command, where i is the input bit number of the trigger input (input bit assignments vary by product—see page 6 for your product's bit assignment configuration). Once defined, an active signal on the specified trigger input will interpolate the current commanded position for all axes. The captured position is interpolated from the last sampled position (of the feedback device selected with the SFB command), the last sampled position error, and the time elapsed since the last sample. *The position sample rate is determined by the SSFR and INDAX commands (system update rate). The accuracy of the position capture is ±50µs x velocity.*

If you issue a PSET (establish absolute position reference) command, any previously captured commanded positions will be offset by the PSET command value.

Example (for the 6250):

```
INFNC25-H       ; Assign trigger input A (TRG-A) as a trigger interrupt input
INFNC26-H       ; Assign trigger input B (TRG-B) as a trigger interrupt input
VAR1=1PCCA      ; Assign captured commanded position on axis 1
                ; (captured when the TRG-A input became active) to variable 1
IF(2PCCB<40)    ; If the captured commanded position on axis 2
                ; (captured when the TRG-B input became active) is
                ; less than 40, do the IF statement
VAR2=1PCCA+10   ; Add 10 to the captured commanded position on axis 1
                ; (captured when the TRG-A input became active) and
                ; assign the sum to variable #2
NIF             ; End IF statement
```

[PCE] Position of Captured Encoder

		Product	Rev
Type	Assignment or Comparison	AT6n00	2/0
Syntax	See below	OEM-AT6n00	n/a
Units	n/a	AT6n50	1.0
Range	n/a	610n	4.0
Default	n/a	615n	1.0
Response	n/a	620n	2.0
See Also	CMDDIR, ENC, ENCPOL, INFNC, [PE], PSET, SCALE, SCLD, SFB, [SS], SSFR, TPCE, TSS	625n	1.0
		6270	1.0

The Position of Captured Encoder (PCE) command is used to assign one of the captured encoder register values (captured when trigger A, B, C or D is activated) to a variable, or to make a comparison against another value. Once the captured encoder register value is assigned to a variable, or a comparison is made, the respective position capture status (reported with bits 25 - 28 in the TSS or SS commands) is cleared, but the position information remains in the register until it is overwritten by a subsequent position capture from the trigger input.

Syntax: VARn=aPCEc where n is the variable number, a is the axis, and c designates trigger A, B, C or D; or [PCE] can be used in an expression such as IF(1PCEB>2345Ø). The PCE command must be used with an axis specifier or it will default to axis 1 (e.g., 1PCEA, 2PCEA, etc.).

The encoder position can be captured only by a trigger input signal (trigger A, B, C or D). The appropriate trigger inputs must be defined as *trigger interrupt* inputs with the INFNCi-H command, where i is the input bit number for the specific trigger input (input bit assignments vary by product—see page 6 for your product's bit assignment configuration).

Steppers: An active trigger input signal from any defined trigger will latch the current encoder positions from all axes and store them in their respective captured encoder arrays. Although the latching may be delayed up to 50 µs from the time the trigger becomes active, all encoder positions are captured within a few microseconds of each other.

If the encoder step mode (ENC1) and scaling (SCALE) are enabled, the captured value is scaled by the distance scaling factor (SCLD). If the encoder step mode is not enabled, the value will be actual encoder counts.

Servos: An active trigger input signal from any defined trigger will capture the current encoder positions from all axes. If encoder feedback is selected with the last SFB command, the captured position is interpolated from the last sampled encoder position and velocity, and the time elapsed since the last sample. *The position sample rate is determined by the SSFR and INDAX commands (system update rate). The accuracy of the position capture is ±50µs x velocity*. If encoder feedback is not selected with the SFB command, the last sampled position is simply stored as the captured position, and the accuracy is one system update period (determined by the SSFR and INDAX commands).

Regardless of the SFB selection, one encoder position is latched in hardware within ±1 encoder count (at max. encoder frequency) when its dedicated trigger input is activated (see table below).

Encoder	AT6250	AT6450	615n*	625n	6270	OEM625n
ENCODER 1	TRG-A	TRG-A	TRG-A	TRG-A	TRG-A	TRG-A
ENCODER 2	TRG-B	TRG-B	TRG-B	TRG-B	n/a	TRG-B
ENCODER 3	TRG-C	TRG-C	n/a	TRG-C	n/a	n/a
ENCODER 4	n/a	TRG-D	n/a	n/a	n/a	n/a

*615n only: TRG-A captures the internal resolver in hardware,
TRG-B captures the external encoder in hardware.

If scaling is enabled (SCALE1), the captured value is scaled by the distance scaling factor (SCLD). If scaling is not enabled, the value will be actual encoder counts. AT6250 & 625n: **ENCODER 3** is never scaled.

NOTE: If you issue a PSET (establish absolute position) command, any previously captured encoder positions will be offset by the PSET command value.

Example (for the AT6400):

```
INFNC26-H       ; Assign trigger input B (TRG-B) as a trigger interrupt input
INFNC25-H       ; Assign trigger input A (TRG-A) as a trigger interrupt input
VAR1=1PCEA      ; Assign captured encoder position on axis 1
                ; (captured when the TRG-A input became active) to variable 1
IF(2PCEB<4000)  ; If the captured encoder count on axis 2 (captured when
                ; the TRG-B input became active) is less than 4000, do the
                ; IF statement
VAR2=1PCEA+4000 ; Add 4,000 to the captured encoder count on axis 1
                ; (captured when the TRG-A input became active) and
                ; assign the sum to variable #2
NIF             ; End IF statement
```

[PCL] Position of Captured LDT Position

Type	Assignment or Comparison
Syntax	See below
Units	n/a
Range	n/a
Default	n/a
Response	n/a
See Also	CMDDIR, INFNC, [LDT], LDTPOL, LDTUPD, PSET, SCALE, SCLD, SFB, [SS], SSFR, TLDT, TPCL, TSS

Product	Rev
AT6n00	n/a
AT6n50	n/a
610n	n/a
615n	n/a
620n	n/a
625n	n/a
6270	1.0

The Position of Captured LDT (PCL) command is used to assign one of the captured LDT position register values (captured when trigger A or B is activated) to a variable, or to make a comparison against another value. Once the captured LDT position register value is assigned to a variable, or a comparison is made, the respective position capture status (reported with bits 25 & 26 in the TSS or SS commands) is cleared, but the position information remains in the register until it is overwritten by a subsequent position capture from the trigger input.

If scaling (SCALE) is enabled, the value assigned to the variable or the value against which the comparison is made is scaled by the distance scaling factor (SCLD). If scaling is not enabled, the value assigned will be actual LDT counts.

Syntax: VARn=aPCLc where n is the variable number, a is the axis, and c designates trigger A or B; or [PCL] can be used in an expression such as IF(1PCLB>25Ø).
The PCL command must be used with an axis specifier or it will default to axis 1 (e.g., 1PCLA, 2PCLB).

The LDT position can be captured only by a trigger input signal (trigger A or B). The appropriate trigger inputs must be defined as *trigger interrupt* inputs with the INFNCi-H command, where i can be 25 or 26, representing trigger inputs A or B, respectively. Once defined, an active signal on the specified trigger input will capture the current LDT position from both axes.

POSITION CAPTURE ACCURACY

If LDT feedback is selected with the SFB command, the captured LDT value is interpolated from the last sampled LDT position and velocity, and the time elapsed since the last sample. *The position sample rate is determined by the SSFR and INDAX commands (system update rate). The accuracy of the position capture is ±50μs x velocity.*

If LDT feedback is NOT selected with the SFB command, the last sampled LDT position is simply stored as the captured LDT position. *The accuracy is one system update period (determined by SSFR and INDAX).*

If you issue a PSET (establish absolute position) command, any previously captured LDT positions will be offset by the PSET value.

Example:

```
INFNC26-H       ; Assign trigger input B (TRG-B) as a trigger interrupt input
INFNC25-H       ; Assign trigger input A (TRG-A) as a trigger interrupt input
VAR1=1PCLA      ; Assign captured LDT position on axis 1 (captured when the
                ; TRG-A input became active) to variable 1
IF(2PCLB<40)    ; If the captured LDT position on axis 2 (captured when the
                ; TRG-B input became active) is less than 40, do the IF
                ; statement
VAR2=1PCLA+10   ; Add 10 to the captured LDT position on axis 1 (captured when
                ; the TRG-A input became active) and assign the sum to
                ; variable #2
NIF             ; End IF statement
```

[PCM] Position of Captured Motor

Type	Assignment or Comparison
Syntax	See below
Units	n/a
Range	n/a
Default	n/a
Response	n/a
See Also	ENC, INFNC, [PCE], PSET, SCALE, SCLD, [SS], TPCM, TSS

Product	Rev
AT6n00	2.0
AT6n50	n/a
610n	4.0
615n	n/a
620n	2.0
625n	n/a
6270	n/a

The Position of Captured Motor (PCM) command is used to assign one of the captured motor register values (captured when trigger A, B, C or D is activated) to a variable, or to make a comparison against another value. Once the captured encoder register value is assigned to a variable, or a comparison is made, the respective position capture status (reported with bits 25 - 28 in the TSS or SS commands) is cleared, but the position information remains in the register until it is overwritten by a subsequent position capture from the trigger input.

If scaling (SCALE) is enabled, the value assigned to the variable or the value against which the comparison is made is scaled by the distance scaling factor (SCLD).

Syntax: VARn=aPCMc where n is the variable number, a is the axis, and c designates trigger A, B, C or D; or [PCM] can be used in an expression such as IF(1PCMB>2345Ø)

The motor position can be captured only by a trigger input signal (trigger A, B, C or D). The appropriate trigger inputs must be defined as *trigger interrupt* inputs with the INFNCi-H command, where i is the input bit number for the specific trigger input (input bit assignments vary by product—see page 6 for your product's bit assignment configuration). Once defined, an active trigger input signal from any defined trigger will latch the current motor positions from all axes and store them in their respective captured motor arrays. Although the latching may be delayed slightly from the time the trigger becomes active (up to 50 μs), all motor positions are captured within a few microseconds of each other.

The PCM command must be used with an axis specifier or it will default to axis 1 (e.g., 1PCMA, 2PCMA, etc.).

NOTE: If you issue a PSET (establish absolute position) command, any previously captured motor positions will be offset by the value specified in the PSET command.

Example (for the AT6400):

```
INFNC26-H       ; Assign trigger input B (TRG-B) as a trigger interrupt input
INFNC25-H       ; Assign trigger input A (TRG-A) as a trigger interrupt input
VAR1=1PCMA      ; Assign captured motor position on axis 1 (captured when
                ; the TRG-A input became active) to variable 1
IF(2PCMB<4000)  ; If the captured motor position on axis 2 (captured when
                ; the TRG-B input became active) is less than 4000, do
                ; the IF statement
VAR2=1PCMA+4000 ; Add 4,000 to the captured motor position on axis 1
                ; (captured when the TRG-A input became active) and
                ; assign the sum to variable #2
NIF             ; End IF statement
```

PCOMP Compile a Profile

		Product	Rev
Type	Compiled Motion; Path Contouring	AT6n00	4.1
Syntax	<!>PCOMP<t>	AT6n50	4.1
Units	t = text (name of program/path)	610n	4.0
Range	Text name of 6 characters or less	615n	4.1
Default	n/a	620n	4.1
Response	n/a	625n	4.1
See Also	DEF, DRES, END, GOBUF, GOWHEN, MEMORY, PA, PAA, PAD, PADA, PAB, PARCOM, PARCOP, PARCM, PARCP, PAXES, PLOOP, PL, PLC, PLIN, PLN, POUTA, POUTB, POUTC, POUTD, PRUN, PSCLD, PUCOMP, PULSE, [SEG], [SS], TDIR, TMEM, TRGFN, TSEG, TSS	6270	4.1

As of revision 4.0, PCOMP was enhanced to compile GOBUF profiles, and contouring is now a standard feature in all multi-axis 6000 products (formerly only the steppers).

More detailed information (including application examples) on multi-axis contours and compiled profiles for individual axes, refer to the Custom Profiling chapter in the *6000 Series Programmer's Guide*.

"Programs" vs. "Compiled Profiles":

- Programs are defined with the DEF and END commands, as demonstrated in the Program Development Scenario in the *Programmer's Guide*.
- Compiled Profiles are defined like programs (using the DEF and END commands), but are compiled with the PCOMP command and executed with the PRUN command. A compiled profile could be a multi-axis contour (a series of arcs and lines), an individual axis profile (a series of GOBUF commands), or a compound profile (combination of multi-axis contours and individual axis profiles).

Compiling and Storing Compiled Paths:

Your controller's memory has two partitions: one for storing programs ("program" memory) and one for storing profiles compiled with the PCOMP command ("compiled" memory). The allocation of memory to these two areas is controlled with the MEMORY command.

Programs intended to be compiled are stored in program memory. After they are compiled with the PCOMP command, they remain in program memory and the segments (see segment command list below) from the compiled profile are stored in compiled memory.

- Contouring segments: PARCM, PARCOM, PARCOP, PARCP, PLIN
- Compiled Motion segments: GOBUF, PLOOP, GOWHEN, TRGFN, POUTA, POUTB, POUTC, POUTD

The TDIR command uses "COMPILED AS A PATH" to denote the programs that are compiled as a compiled profile. TDIR also reports the amount of program storage available, as does the TSEG command. System status bit #29 indicates that compiled memory is 75% full, and system status bit #30 indicates that compiled memory is completely full. (Use TSSF, TSS and [SS] commands to work with system status bits.)

If a compile (PCOMP) fails, system status bit #31 (see TSSF, TSS and [SS] commands) will be set. This status bit is cleared on power-up, reset, or after a successful compile. Possible causes for a failed compile are:

- Errors in profile design (e.g., change direction while at non-zero velocity; distance and velocity equate to < 1 count/system update; preset move profile ends in non-zero velocity).
- Profile will cause a Following error (see TFSF, TFS and [FS] commands).
- Out of memory (see system status bit #30).
- Axis already in motion at the time of a PCOMP command.
- Loop programming errors (e.g., no matching PLOOP or PLN; more than four embedded PLOOP/END loops).

Conditions That Require a Re-Compile:

- If it is desired to change a compiled path's velocity, acceleration, or deceleration, the values must be changed and then the path must be re-compiled.
- If the scaling factors are changed, the program must be downloaded again.
- Compiled Motion ONLY: After compiling (PCOMP) and running (PRUN) a compiled profile, the profile segments will be deleted from compiled memory if you cycle power or issue a RESET command.

IMPORTANT NOTES

CONTOURING: The mechanical resolution of all axes used for contouring (specified with the PAXES command) must be identical; scaling cannot compensate for mechanical variances in resolution. In addition, all contouring axes must have the same pulse width (PULSE), and the same drive resolution (DRES) settings (steppers) or the same number of feedback device counts per unit of linear travel (servos). If you change the PULSE setting, you will need to recompile (PCOMP) any previously compiled paths.

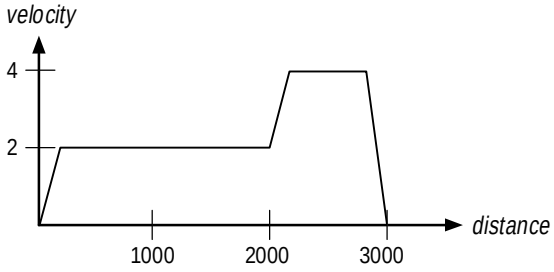
COMPILED MOTION: When using compiled loops (PLOOP and PLN), the last segment within the loop must end at zero velocity or there must be a final GOBUF segment placed outside the loop. Otherwise an error will result when the profile is compiled. The error is "ERROR: MOTION ENDS IN NON-ZERO VELOCITY-AXIS n".

CONTOURING EXAMPLE

```
DEF prog1       ; Begin definition of program named prog1
PAXES1,2,3,4    ; Set axes 1, 2, 3, & 4 as the X, Y, Tangent, &
                ; Proportional axes, respectively
PPRO2.25        ; Proportional axis path ratio = 2.25
; ***********************************************
; * Put                                         *
; * MULTIPLE MOTION SEGMENT DEFINITIONS         *
; * Here                                        *
; ***********************************************
END             ; End definition of path prog1
PCOMP prog1     ; Compile path prog1
PRUN prog1      ; Execute path prog1
```

COMPILED MOTION EXAMPLE (see profile below)

```
DEF prog2       ; Begin definition of program named prog2
A10,10          ; Set A, V, and D values for axes 1 and 2
V2,2
D2000,2000
GOBUF11         ; First segment of motion for axes 1 and 2
V4,4            ; New A,V, and D values
AD50,50
D1000,1000
GOBUF11         ; Second segment
END             ; End definition of prog2
PCOMP prog2     ; Compile prog2
PRUN prog2      ; Execute prog2
```



[PE] Position of Encoder

Type	Assignment or Comparison
Syntax	See below
Units	(see description below)
Range	n/a
Default	n/a
Response	n/a
See Also	CMDDIR, CNTE, ENC, ENCPOL, [FB], GOWHEN, INFNC, [PC], [PCE], [PER], [PM], PSET, SCALE, SCLD, SFB, TFB, TPE

Product	Rev
AT6n00	1.0
OEM-AT6n00	n/a
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Position of Encoder (PE) command is used to assign one of the encoder register values to a variable, or to make a comparison against another value.

Steppers: The encoder value is scaled by the distance scaling factor (SCLD), if encoder step mode (ENC1) and scaling (SCALE) are enabled. If the encoder step mode is not enabled, the value will be actual encoder counts. If the encoder channel has been defined as a counter input (CNTE), then the PE command will report a reading of zero for that specific encoder channel.

Servos: If scaling is enabled (SCALE1), the encoder/resolver value is scaled by the distance scaling factor (SCLD). If scaling is not enabled, the value assigned will be actual encoder/resolver counts. AT6250 & 625n: **ENCODER 3** is never scaled. 615n: The position of the internal resolver is referenced as axis 1; the position of the external encoder is referenced as axis 2.

If you issue a PSET command, the encoder/resolver position value will be offset by the PSET command value.

Syntax: VARn=aPE where n is the variable number, and a is the axis, or [PE] can be used in an expression such as IF(1PE>2345Ø). The PE command must be used with an axis specifier or it will default to axis 1 (e.g., 1PE, 2PE, etc.).

Example:

```
VAR1=1PE         ; Encoder/resolver position for axis 1 is assigned to variable 1
IF(2PE<4000)     ; If the encoder/resolver count for axis 2 is less than 4000,
                 ; do the IF statement
VAR2=3PE+4000    ; Encoder/resolver position for axis 3 plus 4000 is assigned
                 ; to variable 2
NIF              ; End IF statement
```

[PER] Position Error

Type	Assignment or Comparison
Syntax	See below
Units	n/a
Range	n/a
Default	n/a
Response	n/a
See Also	ANIPOL, CMDDIR, DRES, ENCPOL, ERES, LDTPOL, SCLD, SFB, SMPER, TAS, TPER, TPE, TPC

Product	Rev
AT6n00	1.4
OEM-AT6n00	n/a
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.5
625n	1.0
6270	1.0

The Position Error (PER) command is used to assign the current position error of each axis to a variable, or to make a comparison against another value. The value assigned to the variable or the value against which the comparison is made is measured in feedback device counts and is scaled by the distance scaling factor (SCLD), if scaling is enabled with the SCALE1 command.

Steppers: This command can be used **only** when the encoder mode (ENC1) is enabled.

Servos: The position error is the difference between the commanded position and the actual position read by the feedback device. This error is calculated every sample period and can be displayed at any time using the TPER command.

Syntax: VARn=aPER where n is the variable number, and a is the axis, or [PER] can be used in an expression such as IF(1PER>5Ø). The PER command must be used with an axis specifier or it will default to axis 1 (e.g., 1PER, 2PER, etc.).

Example:
```
VAR1=1PER       ; Position error for axis 1 is assigned to variable 1
IF(2PER>2000)   ; If the position error for axis 2 is >2000 encoder counts,
                ; do the IF statement (enable output #4)
OUTXXX1         ; Enable output #4
NIF             ; End IF statement
```

[PI] PI (π)

		Product	Rev
Type	Operator (Trigonometric)	AT6n00	1.0
Syntax	See examples below	AT6n50	1.0
Units	n/a	610n	4.0
Range	n/a	615n	1.0
Default	n/a	620n	1.0
Response	n/a	625n	1.0
See Also	[=], [+], [-], [*], [/], [&], [|], [^], [~], [ATAN], [COS], IF, [SIN], [SQRT], [TAN], VAR	6270	1.0

The (PI) command is assigned the value 3.14159265. There are 2π radians in 360°. This command is useful for doing trigonometric functions in radian units (RADIAN command).

Example:
```
VAR1=PI         ; 3.14159265 is assigned to variable 1
VAR2=2 * PI     ; 2 pi is assigned to variable 2
```

PL Define Path Local Mode

		Product	Rev
Type	Path Contouring	AT6n00	1.0
Syntax	<!>PL<b>	AT6n50	4.1
Units	n/a	610n	n/a
Range	b = 0 (work coordinates) or 1 (local coordinates)	615n	n/a
Default	0	620n	1.0
Response	No response - Must be defining a path (DEF)	625n	4.1
See Also	PAB, PLC, PWC	6270	4.1

The Define Path Local Mode (PL) command is used to specify the use of either the Local coordinate system or the Work coordinate system. Endpoints are allowed to be specified as absolute positions, and these positions may either be in the Work or the Local coordinate system. Programming may switch between Local and Work coordinates before any segment or group of segments.

When switching to Local coordinates, the starting coordinates of the next segment in the Local coordinate system must be specified with the PLC command before the PL1 command is issued.

When using the Work coordinate system (PLØ), the starting coordinates of the next segment in the Work coordinate system may be specified with the PWC command for the purpose of shifting the Work coordinate system. If the PWC command is not given, the previous Work coordinate system is used.

Example:

```
PSCLA25000      ; Set path acceleration scale factor to 25000 steps/unit/unit
PSCLD25000      ; Set path distance scale factor to 25000 steps/unit
PSCLV25000      ; Set path velocity scale factor to 25000 steps/unit
SCALE1          ; Enable scaling factor
PV5             ; Set path velocity to 5 units/sec
PA50            ; Set path acceleration to 50 units/sec/sec
PAD100          ; Set path deceleration to 100 units/sec/sec
DEF prog1       ; Begin definition of path named prog1
PAXES1,2        ; Set axes 1 and 2 as the X and Y contouring axes
PAB1            ; Set to absolute coordinates
PWC0,0          ; Specify X and Y data, work coordinates
PL0             ; Specify work coordinate system
PLIN1,1         ; Specify X-Y endpoint position to create a 45 degree angle
                ; line segment
PLC0,0          ; Specify X and Y data, local coordinates
PL1             ; Specify local coordinate system
PARCOP0,0,5,0   ; Specify incremental X-Y endpoint position and X-Y center
                ; position for full circle clockwise arc
PLIN0,11        ; Specify X-Y endpoint position to create a 90 degree angle
                ; line segment
PLC0,0          ; Specify X and Y data, local coordinates
PL1             ; Specify local coordinate system
PARCOP0,0,5,0   ; Specify incremental X-Y endpoint position and X-Y center
                ; position for full circle clockwise arc
PL0             ; Specify work coordinate system
PLIN0,0         ; Specify X-Y endpoint position to create a line segment back
                ; to 0,0
END             ; End definition of path prog1
PCOMP prog1     ; Compile path prog1
PRUN prog1      ; Execute path prog1
```

PLC — Define Path Local Coordinates

Type	Path Contouring
Syntax	<!>PLC<r>,<r>
Units	r = units
Range	0.00000 - ±999,999,999
Default	n/a
Response	No response - Must be defining a path (DEF)
See Also	PAB, PL, PSCLD, PWC, SCALE

Product	Rev
AT6n00	1.0
AT6n50	4.1
610n	n/a
615n	n/a
620n	1.0
625n	4.1
6270	4.1

The Define Path Local Coordinates (PLC) command is used to specify the Local X -Y coordinate data required for subsequent segment definition in the Local coordinate system. This command places the X -Y coordinate value of the Local coordinate system at the beginning of the next segment. (The first <r> is the X coordinate, the second <r> is the Y coordinate.) This command must be used before the PL1 command is given.

Both position values are expressed in terms of motor steps, regardless of the current ENC command setting.

Scaling: If scaling (SCALE) is enabled, the PLC command values entered are internally multiplied by the path distance scaling factor (PSCLD) to convert user units to motor steps. The distance values may be truncated if the values entered exceed the distance resolution at the given scaling factor. For further discussion on path distance scaling, refer to the PSCLD command description.

Example: Refer to Define Path Local Mode (PL) command example.

PLIN — Move in a Line

		Product	Rev
Type	Path Contouring	AT6n00	1.0
Syntax	<!><@>PLIN<r>,<r>	AT6n50	4.1
Units	r = units	610n	n/a
Range	0.00000 - ±999,999,999	615n	n/a
Default	n/a	620n	1.0
Response	No response - Must be defining a path (DEF)	625n	4.1
See Also	PAB, PL, PLC, PSCLD, PWC, SCALE	6270	4.1

The Define Line Segment (PLIN) command is used to specify a line segment. The placement, length, and orientation of the line are completely specified by the endpoint of the line segment and the endpoint of the previous segment (current position). Segment endpoint position specifications may be either absolute (PAB1) with respect to the user defined coordinate system, or incremental (PABØ), relative to the start of each individual segment.

When the PLIN command is received, the first value is taken as the X endpoint coordinate and the second value is taken as the Y endpoint coordinate.

Both position values are expressed in terms of motor steps, regardless of the current ENC command setting.

Scaling: If scaling (SCALE) is enabled, the PLIN command values entered are internally multiplied by the path distance scaling factor (PSCLD) to convert user units to motor steps. The distance values may be truncated if the values entered exceed the distance resolution at the given scaling factor. For further discussion on path distance scaling, refer to the PSCLD command description.

Example: Refer to Define Path Local Mode (PL) command example.

PLN — Loop End, Compiled Motion

		Product	Rev
Type	Compiled Motion	AT6n00	4.1
Syntax	<@>PLN<b><b><b><b>	AT6n50	4.1
Units	n/a	610n	4.0
Range	b = 1 (end loop), 0 or X (don't end loop)	615n	4.1
Default	n/a	620n	4.1
Response	No response; instead ends loop for compiled motion	625n	4.1
See Also	GOBUF, PCOMP, PLOOP, PRUN, PUCOMP	6270	4.1

The Loop End, Compiled Motion (PLN) command specifies the end of an axis-specific compiled motion profile loop, as initiated with the PLOOP command.

Programming Example: see PLOOP.

PLOOP — Loop Start, Compiled Motion

Type	Compiled Motion
Syntax	<@>PLOOP<i>,<i>,<i>,<i>
Units	i = designated number of loops for specified axis
Range	0-2,147,483,647 (2^{31}-1) 0 = infinite loop
Default	n/a
Response	No response; instead starts loop for compiled motion
See Also	GOBUF, PCOMP, PLN, PRUN, PUCOMP

Product	Rev
AT6n00	4.1
AT6n50	4.1
610n	4.0
615n	4.1
620n	4.1
625n	4.1
6270	4.1

The PLOOP command specifies the beginning of an axis-specific profile loop. All subsequent segments defined before the PLN command are included within that loop. The number in a given axis field specifies the number of loops to be executed for that axis. If that number is a zero or blank, then the loop will be executed infinitely. The PLOOP command can be nested up to four levels deep within a program.

When using compiled loops (PLOOP and PLN), the last segment within the loop must end at zero velocity or there must be a final GOBUF segment placed outside (after) the loop. Otherwise an error will result when the profile is compiled. The error is "ERROR: MOTION ENDS IN NON-ZERO VELOCITY-AXIS n".

The PLOOP command will consume one segment of compiled space.

Example:

```
DEF prog1       ; Begin definition of prog1
V1              ; Set velocity to 1 unit/sec
D1000           ; Set distance to 1000 units
GOBUF1          ; Segment of motion sent to buffer

PLOOP3          ; Start loop of the subsequent move profile

V10             ; Set velocity to 10 units/sec
D25000          ; Set distance to 25000 units
GOBUF1          ; First segment within loop sent to buffer

V2              ; Set velocity to 2 units/sec
D1000           ; Set distance to 1000 units
GOBUF1          ; Second segment of motion within loop sent to buffer

V1              ; Set velocity to 1 unit/sec
D25000          ; Set distance to 25000 units
GOBUF1          ; Third segment within loop sent to buffer

PLN1            ; Close loop

V.5             ; Set velocity to 0.5 units/sec
D100            ; Set distance to 100 units
GOBUF1          ; Segment of motion sent to buffer (outside loop)

END             ; End definition of prog1

PCOMP prog1     ; Compile prog1
PRUN prog1      ; Execute prog1
```

[PM] — Position of Motor

Type	Assignment or Comparison
Syntax	See below
Units	n/a
Range	n/a
Default	n/a
Response	n/a
See Also	GOWHEN, [PE], PSET, SCALE, SCLD, TPM

Product	Rev
AT6n00	1.0
AT6n50	n/a
610n	4.0
615n	n/a
620n	1.0
625n	n/a
6270	n/a

The Position of Motor (PM) command is used to assign one of the motor position register values to a variable, or to make a comparison against another value. The value assigned to the variable or the value

against which the comparison is made is scaled by the distance scaling factor (SCLD), if scaling is enabled (SCALE1). If scaling is not enabled, the value is in steps.

Syntax: VARn=aPM where n is the variable number, and a is the axis, or [PM] can be used in an expression such as IF(1PM>2345Ø). The PM command must be used with an axis specifier or it will default to axis 1 (e.g., 1PM, 2PM, etc.).

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

Example:
```
VAR1=1PM        ; Motor position for axis 1 is assigned to variable 1
IF(2PM<4000)    ; If the motor position for axis 2 is less than 4000,
                ; do the IF statement
VAR2=3PM+4000   ; Motor position for axis 3 plus 4000 is assigned to variable 2
NIF             ; End IF statement
```

[PMAS] Current Master Cycle Position

		Product	Rev
Type	Following and Assignment or Comparison	AT6n00	3.0
Syntax	See below	AT6n50	3.0
Units	n/a	610n	4.0
Range	n/a	615n	3.0
Default	n/a	620n	3.0
Response	n/a	625n	3.0
See Also	ENC, FMCNEW, FMCP, FOLMAS, FOLMD, [FS], GOWHEN, SCALE, SCLMAS, TPMAS, TFS	6270	3.0

The Position of Master (PMAS) command is used to assign the master position register value to a variable, or to make a comparison against another value. This value may be used for subsequent decision making, or for recording the cycle position corresponding to some other event.

PMAS is unique among position assignment variables, because its value rolls over to zero each time the entire master cycle length (FMCLEN value) has been traveled. If it is desired to WAIT or GOWHEN on a master cycle position of the next master cycle, one master cycle length (value of FMCLEN) should be added to the master cycle position specified in the argument. This allows commands that sequence slave events through a master cycle to be placed in a loop. The WAIT or GOWHEN command at the top of the loop could execute, even though the actual master travel had not finished the previous cycle. This is done to allow a PMAS value which is equal to the master cycle length to be specified and reliably detected. When using PMAS with IF, UNTIL, or WHILE arguments, the instantaneous PMAS value is used. Be careful to avoid specifying PMAS values that are nearly equal to the master cycle length (FMCLEN), because rollover may occur before a PMAS sample is read.

The master must be assigned first (FOLMAS command) before this command will be useful.

If scaling is enabled (SCALE1), the PMAS value is scaled by the master scaling factor (SCLMAS). If scaling is disabled (SCALEØ), the PMAS value is in counts.

Syntax: VARn=aPMAS where n is the variable number and a is the axis number, or [PMAS] can be used in an expression such as IF(2PMAS>2345Ø). The PMAS command must be used with an axis specifier, or it will default to axis 1 (e.g., VAR1=1PMAS, IF(2PMAS>5ØØ), etc.).

Example: (refer also to FOLEN example #2)
```
IF(2PMAS>4.3)   ; If the master for axis 2 has traveled more than 4.3
                ; master user units then do the IF statement
OUT.12=b1       ; Set output #12 to 1
NIF             ; End of IF statement
VAR14=1PMAS     ; Set VAR14 to axis 1's master cycle position
```

PORT — Designate Destination COM Port

		Product	Rev
Type	Communication Interface	AT6n00	n/a
Syntax	<!>PORT<i>	AT6n50	n/a
Units	i = port number	610n	4.0
Range	1 (COM1), 2 (COM2)	615n	4.1
Default	1	620n	4.1
Response	n/a	625n	4.1
See Also	], [, BOT, DRPCHK, E, EOL, EOT, ERRDEF, ERRLVL, ERROK, ERRBAD, [READ], WRITE, XONXOFF	6270	4.1

The Designate Destination Port (PORT) command is used to determine which COM port is affected by the DRPCHK, E, ECHO, BOT, EOL, EOT, ERROK, ERRBAD, ERRDEF, ERRLVL, and XONOFF commands. It also specifies the port to which responses and prompts from stored programs should be sent.

The PORT command also selects the target port through which the WRITE and READ commands transmit ASCII text strings. The DWRITE command (as well as all other RP240 commands) will affect the RP240 regardless of the PORT command setting. If no RP240 is detected, the commands are sent to the COM2 port. DWRITE text strings are always terminated with a carriage return.

Example (The PORT command can be used to designate EOT parameters for both ports. Assume that port COM1 is being used to communicate to the controller.)

```
PORT1             ; Select COM1 for EOT setup
EOT45,49,10       ; EOT for COM1 is -1<lf>
TPE               ; Send “Transfer Position of Encoder” response to COM1
                  ; using EOT 45,49,10
PORT2             ; Select COM2 for EOT setup
EOT45,50,10       ; EOT for COM2 is -2<lf>
TPM               ; Send “Transfer Position of Motor” response to COM1
                  ; using EOT 45,49,10
```

Example (The PORT command specifies both port setups and response destinations in a stored program.)

```
DEF qwe           ; Begin definition of qwe
PORT1
EOT45,49,10       ; EOT for COM1 is -1<lf>
TPE               ; Send “Transfer Position of Encoder” response to COM1
                  ; using EOT 45,49,10
PORT2
EOT45,50,10       ; EOT for COM2 is -2<lf>
TPM               ; Send "Transfer Position of Motor" response to COM2
                  ; using EOT 45,50,10
END               ; End definition of qwe
```

POUT — Compiled Output

		Product	Rev
Type	Path Contouring; Compiled Motion	AT6n00	4.1
Syntax	<!>POUT<n><b><b>...<b><b> (16 bits)	AT6n50	4.1
Units	n = axis identifier letter (for compiled motion only); b = enable bit	610n	4.0
Range	n = A-D for axes 1-4, respectively (for compiled motion only); b = 0 (off), 1 (on), or X (don't change)	615n	4.1
Default	0	620n	4.1
Response	n/a	625n	4.1
See Also	GUBUF, OUT, OUTEN, OUTFNC, OUTLVL, PCOMP, PRUN, PUCOMP	6270	4.1

As of revision 4.0, contouring was made a standard feature in all multi-axis 6000 products (formerly only the steppers). In addition, POUT was modified to accommodate Compiled Motion Profiling; when using POUT with compiled motion, you must add an axis identifier — the “n” in the syntax is for the letter designator for the axis (A-D for axes 1-4, respectively):

Contouring: POUT

Compiled Motion: POUTA (apply output pattern to the profile for axis #1)
POUTB (apply output pattern to the profile for axis #2)
POUTC (apply output pattern to the profile for axis #3)
POUTD (apply output pattern to the profile for axis #4)

POUT controls the first 16 programmable outputs defined as "Programmable Outputs". "Programmable Output" is the default function assigned to all programmable outputs; the function assignment is determined by the OUTFNC command, OUTFNCi-A assigns the "Programmable Output" function. For example, if only outputs 3 and 5 were defined as "programmable" (outputs 1, 2, and 4 had some other OUTFNC function assignment), then turning on only outputs 3 and 5 for axis 1 would require the command POUTA11, not POUTAxx1x1.

If you wish to set only one output value, instead of all outputs, use the bit select (.) operator, followed by the number of the specified output. For example, POUT.12-1 turns on only output 12 (for contouring) and POUTA.12-1 turns on only output 12 for the axis 1 profile.

The POUT command consumes one segment of compiled memory.

The programmable outputs are sampled once per "system update". The system update for stepper products is 2 ms; for servos it depends on the current SSFR and INDAX settings (see table in SSFR command description).

Contouring ONLY:

The POUT command specifies the programmable output bit pattern to be applied to the outputs at the beginning of the next segment and remain throughout that segment. The POUT command may be issued before any segment definition command, and will affect all subsequent segments until a new POUT command is issued. A POUT command will not take affect if there is no segment definition command following it. To change the programmable outputs at the end of a path, the standard output (OUT) command must be used after the path is executed. These segment defined output patterns are stored as part of the compiled path definition.

CONTOURING EXAMPLE: Refer to the PARCOM command example.

COMPILED MOTION EXAMPLES:

```
OUTFNC3-A       ; Default output function
OUTFNC6-A       ; Default output function
DEF P1          ; Define program P1
D1000,25000     ; Set distance to travel
GOBUF11         ; Motion segments for axes 1 and 2
POUTA.3-1       ; Turn on output 3 when axis 1 travels to 1000 steps
D2000,50000     ; New distance commanded
GOBUF11         ; Motion segments for axes 1 and 2
POUTA.3-0       ; Turn off output 3 when axis 1 travels 2000 additional steps
POUTB.6-1       ; Turn on output 6 when axis 2 travels to 75000 steps
D1000,25000     ; New distance commanded
GOBUF11         ; Motion segment for axes 1 and 2
POUTB.6-0       ; Turn off output 6 when axis 2 travels 25000 additional steps
END             ; End program definition

PCOMP P1        ; Compiled program P1
PRUN P1         ; Execute program P1
```

When executing a Compiled Following profile, the POUTn statement is always executed as programmed. Therefore, in order to make sure an output is on for a given motion segment no matter what direction the master is traveling, you should use two POUTn statements (see example below).

```
POUTA.3-0       ; Turn off output 3 for axis 1 - master going backwards
POUTA.3-1       ; Turn on output 3 for axis 1 - master going forwards
GOBUF1          ; Motion segments for axes 1
POUTA.3-1       ; Turn on output 3 for axis 1 - master going backwards
POUTA.3-0       ; Turn off output 3 for axis 1 - master going forwards
```

If you desire to "pulse" an output (turn on for a given amount of time), then use the POUTn command along with the GOWHEN(T=n) command. For example:

```
POUTA.1-1       ; Turn on output 1
GOWHEN(T=120)   ; Wait for 120 milliseconds
POUTA.1-0       ; Turn off output 1
```

PPRO Path Proportional Axis

Type	Path Contouring
Syntax	<!>PPRO<r>
Units	r = ratio value
Range	±0.001 - 1000.000
Default	n/a
Response	No response - Must be defining a path (DEF)
See Also	PAXES

Product	Rev
AT6200	n/a
AT6400	1.0
AT6250	n/a
AT6450	4.1
610n	n/a
615n	n/a
620n	n/a
625n	n/a
6270	n/a

The Path Proportional Axis (PPRO) command is used to specify the proportional axis to path travel ratio. The proportional axis will keep a position that is proportional to the distance traveled along the X-Y path as the path is executed. This allows the proportional axis to act as the Z axis in helical interpolation or to control the motion of any object which moves with distance and velocity proportional to the path.

The PPRO command should be given prior to any contour segments during a path definition. A negative value for the proportional axis ratio simply causes motion in the negative direction as path travel in the X-Y plane gets larger.

Example: (see contouring programming example in the PRUN command description)

PRTOL Path Radius Tolerance

Type	Path Contouring
Syntax	<!>PRTOL<r>
Units	r = allowable radius error
Range	0.00001 - ±999,999,999
Default	1
Response	No response - Must be defining a path (DEF)
See Also	PARCM, PARCOM, PARCOP, PARCP, PSCLD, SCALE

Product	Rev
AT6n00	1.0
AT6n50	4.1
610n	n/a
615n	n/a
620n	1.0
625n	4.1
6270	4.1

The Path Radius Tolerance (PRTOL) command is used to specify the allowable radius error that is encountered when contouring.

The radius error is encountered in one of two ways. The first way is through use of the PARCM or PARCP commands. This error is the difference between the radius value specified in the PARCM or PARCP command and the minimum radius implied by the starting point and endpoint. If the radius provided in the command is smaller than the minimum radius implied by the distance from starting to endpoints and the error is within the radius tolerance then just enough is added to the radius to make a half circle.

A second way to encounter a radius tolerance error is with the PARCOM or PARCOP commands. This error is the difference between the radius implied by the start point and center point and the radius implied by the end point and center point. If the difference in the two radius values is within the radius tolerance specified, then the center point is moved such that an arc can be traveled through the start point and endpoint. The PRTOL command can be executed many times within a path definition allowing some arcs to be exactly known and others to be approximated.

If the radius error exceeds the PRTOL value, an error message is sent.

Steppers only: The PRTOL radius error value is expressed in terms of motor steps, regardless of the current ENC command setting.

Scaling: If scaling (SCALE) is enabled, the PRTOL command values entered are internally multiplied by the path distance scaling factor (PSCLD) to convert user units to motor steps. The distance values may be truncated if the values entered exceed the distance resolution at the given scaling factor. For further discussion on path distance scaling, refer to the PSCLD command description.

Example:

```
PSCLA25000      ; Set path acceleration scale factor to 25000 steps/unit/unit
PSCLD25000      ; Set path distance scale factor to 25000 steps/unit
PSCLV25000      ; Set path velocity scale factor to 25000 steps/unit
SCALE1          ; Enable scaling factor
PV5             ; Set path velocity to 5 units/sec
PA50            ; Set path acceleration to 50 units/sec/sec
PAD100          ; Set path deceleration to 100 units/sec/sec
DEF prog1       ; Begin definition of path named prog1
PAXES1,2        ; Set axes 1 and 2 as the X and Y contouring axes
PAB0            ; Set to incremental coordinates
PRTOL0.001      ; Allow 25 steps (0.001 x 25000) radius error
PARCM5,5,5      ; Specify incremental X-Y endpoint position and radius
                ; arc <180 degree for quarter circle counter-clockwise arc
PARCP5,-5,-5    ; Specify incremental X-Y endpoint position and radius
                ; arc >180 degree for three quarter circle clockwise arc
END             ; End definition of path prog1
PCOMP prog1     ; Compile path prog1
PRUN prog1      ; Execute path prog1
```

PRUN — Run a Compiled Profile

Type	Compiled Motion; Path Contouring
Syntax	<!>PRUN<t>
Units	t = text (name of path program)
Range	text name of 6 characters or less
Default	n/a
Response	n/a
See Also	DEF, END, PCOMP, PUCOMP, GOBUF, PLOOP, PLN, COMEXC

Product	Rev
AT6n00	4.1
AT6n50	4.1
610n	4.0
615n	4.1
620n	4.1
625n	4.1
6270	4.1

As of revision 4.0, contouring was made a standard feature in all multi-axis 6000 products (formerly only the steppers). In addition, PRUN was modified to accommodate executing Compiled Motion Profiles.

The Run a Pre-Compiled Program (PRUN) command is used to start execution of a previously compiled program. All the required information about the program or path whose name is specified in the PRUN command has already been stored by the definition commands (DEF and END)and compiled by the PCOMP command. If any of the axes included in the specified path are not ready, the path will not be executed. An axis is not ready if it is shutdown, moving, or in joystick mode. When execution of a pre-compiled program begins, all included axes become busy until motion has completed. COMEXC1 mode must be enabled in order for command processing to continue once a motion invoking command has been initiated with PRUN. If you use the PRUN command within a program while in COMEXC1 mode, it functions as a GO and returns control back to the original program after the embedded program's motion is started (control is returned to the first command immediately following the PRUN command). If in COMEXCØ mode, command processing will not continue until the motion invoking command has completed its movement.

CONTOURING EXAMPLE:

```
DEL prog1         ; Delete prog1
DEF prog1         ; Begin definition of path named prog1
PAXES1,2,3,4      ; Set axes 1,2,3,4 as the X, Y, Tangent, and
                  ; Proportional axes respectively
PPRO2.25          ; Proportional axis path ratio = 2.25
;  ****************************************
;  *  Add multiple path segment           *
;  *  definitions in this                 *
;  *  portion of the                      *
;  *  program                             *
;  ****************************************
END               ; End definition of path prog1
PCOMP prog1       ; Compile path prog1
PRUN prog1        ; Execute path prog1
```

COMPILED MOTION EXAMPLE:

```
@D25000          ; Set distance parameter for all axes
DEL prog1        ; Delete prog1
DEF prog1        ; Define prog1
PLIN1000,1000    ; Line segment on axis 1 and 2
GOBUFxx11        ; Compiled motion on axis 3 and 4
END              ; End definition of prog1
PCOMP prog1      ; Compile prog1
PRUN prog1       ; Execute prog1
TPM              ; Check position of motors for stepper products
                 ; (or, if using a servo product, use TPC to check
                 ; commanded position).  A sample response would be:
                 ; "*TPM1000,1000,25000,25000"
```

PS — Pause Program Execution

Type	Program Flow Control
Syntax	<!>PS
Units	n/a
Range	n/a
Default	n/a
Response	n/a
See Also	C, COMEXR, COMEXS, K, S, [SS], TSS

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Pause Program Execution (PS) command pauses execution of commands in the command buffer. If a PS command is executed, no commands after the PS will be executed until a !C command is received. However, additional commands may still be placed in the command buffer.

The PS command does not pause motion. In order for motion to be paused, the S and the COMEXS commands should be used.

Example:

```
PS               ; Stop execution of command buffer until !C command
MA0XXX           ; Incremental mode for axis 1
D10000           ; Set distance to 10000 units on axis 1
GO1000           ; Initiate motion on axis 1
D,20000          ; Set distance to 20000 units on axis 2
GO0100           ; Initiate motion on axis 2
; ********************************************************************
; * NOTE:                                                            *
; * No commands after the PS command will be executed until a !C     *
; * command is received.                                             *
; ********************************************************************
```

PSCLA — Path Acceleration Scale Factor

Type	Scaling; Path Contouring or Motion (Linear Interpolated)
Syntax	<!>PSCLA<i>
Units	i = steps/unit
Range	1 - 999,999
Default	Steppers: 25000 Servos: Depends on feedback source for axis #1 (4000 if encoder, 819 if ANI, 432 if LDT)
Response	PSCLA: *PSCLA25000
See Also	PA, PAA, PAD, PADA, PSCLD, PSCLV, SCALE, SFB

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	n/a
615n	n/a
620n	1.0
625n	1.0
6270	1.0

When scaling is enabled (SCALE1), all path acceleration (PA and PAA) and path deceleration (PAD and PADA) values are internally multiplied by the Path Acceleration Scale Factor (PSCLA) value. Since the units are steps/unit, and all the acceleration values are in units/sec/sec, all accelerations will thus be internally represented as steps/sec/sec. The Path Acceleration Scale Factor (PSCLA) command will not scale the accel/decel values unless the scaling is enabled (SCALE1).

Steppers: The entered values are always in reference to motor steps, not encoder steps, regardless of the ENC command setting. If scaling is enabled (SCALE1), the entered acceleration and deceleration values are internally multiplied by the path acceleration scaling factor (PSCLA) to convert user units/sec/sec to motor steps/sec/sec. If scaling is not enabled (SCALEØ), the path acceleration and deceleration values are entered in motor revs/sec/sec; these values are internally multiplied by the drive resolution (DRES) value to obtain acceleration and deceleration values in motor steps/sec/sec for the motion trajectory calculations.

Servos: If scaling is enabled (SCALE1), the entered PA/PAA and PAD/PADA values are internally multiplied by the path acceleration scaling factor (PSCLA) to convert user units/sec/sec to encoder, LDT, or ANI steps/sec/sec. If scaling is not enabled (SCALEØ), the path accel and decel values are entered in encoder or resolver revs/sec/sec, LDT inches/sec/sec, or ANI volts/sec/sec. Encoder/resolver and LDT values are internally multiplied by the encoder resolution (ERES) value or the LDT resolution (LDTRES) to obtain accel/decel values in steps/sec/sec for the motion trajectory calculations.

The path acceleration and deceleration remain set until you change them with a subsequent path acceleration and deceleration (PA/PAA & PAD/PADA) commands. Decelerations outside the valid range are flagged as an error, with a message *INVALID DATA-FIELD x, where x is the field number. When an invalid accel or decel is entered the previous accel or decel value is retained.

As the path acceleration scaling factor (PSCLA) changes, the resolution of the path accel/decel values and the number of positions to the right of the decimal point also change (see table at right). An accel or decel value with greater resolution than allowed will be truncated. For example, if scaling is set to PSCLA1Ø, the PA9.9999 command would be truncated to PA9.9.

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

The following equations can help you determine the range of path accel and decel values.

Product	**Min. Path Accel or Decel (resolution)**		**Max. Path Accel or Decel** (Servos: determined by the feedback source selected for axis 1)	
Steppers	$\frac{0.001 \times \text{DRES}}{\text{PSCLA}}$		$\frac{999.9999 \times \text{DRES}}{\text{PSCLA}}$	
Servos	Encoder Feedback:	$\frac{0.001 * \text{ERES}}{\text{PSCLA}}$	Encoder Feedback:	$\frac{999.9999 * \text{ERES}}{\text{PSCLA}}$
	LDT Feedback:	$\frac{0.001 * \text{LDTRES}}{\text{PSCLA}}$	LDT Feedback:	$\frac{999.9999 * \text{LDTRES}}{\text{PSCLA}}$
	ANI Feedback:	$\frac{0.819}{\text{PSCLA}}$	ANI Feedback:	$\frac{818999.9181}{\text{PSCLA}}$

NOTE

If scaling is desired for a particular path, scaling must be enabled (SCALE1) and all path scaling factors (PSCLA, PSCLD, PSCLV) must be specified prior to defining the path. Scaling cannot be enabled and scaling factors cannot be specified within a path definition.

Example: Refer to the Path Acceleration (PA) command example.

PSCLD Path Distance Scale Factor

		Product	Rev
Type	Scaling; Path Contouring	AT6n00	1.0
Syntax	<!>PSCLD<i>	AT6n50	4.1
Units	i = steps/unit	610n	n/a
Range	1 - 999,999	615n	n/a
Default	1	620n	1.0
Response	PSCLD: *PSCLD1	625n	4.1
See Also	PARCM, PARCOM, PARCOP, PARCP, PLC, PLIN, PRTOL, PSCLA, PSCLV, PWC, SCALE	6270	4.1

When scaling is enabled (SCALE1), all distance (PARCM, PARCOM, PARCOP, PARCP, PLC, PLIN, PRTOL, PWC) values are internally multiplied by the Path Distance Scale Factor (PSCLD) value. Since the units are steps/unit, all distances will thus be internally represented in steps. The PSCLD command will not scale a commanded distance unless the Scale command is enabled (SCALE1). If the Scale (SCALE) command is not enabled, all distance values are in steps.

This command is useful for specifying path or linear interpolated move distances in any unit. (e.g., Given a 25000 step/rev drive and wanting distance units in revs, then PSCLD would be set to 25000.)

As the path distance scaling factor (PSCLD) changes, the resolution of the distance values entered and the number of positions to the right of the decimal point also change. For instance, if scaling is set to PSCLD25000, the PARCOP55.99999,22.88671,37.86752,21.11112 command would be truncated to PARCOP55.9999,22.8867,37.8675,21.1111. 6270 only: In the table below, shift the decimal place in the Path Distance Range column one place to the left.

PSCLD (steps/unit)	Path Distance Resolution (units)	Path Distance Range (units)	Decimal Places
1 - 9	1	0 - ±999,999,999	0
10 - 99	0.1	0.0 - ±99,999,999.9	1
100 - 999	0.01	0.00 - ±9,999,999.99	2
1000 - 9999	0.001	0.000 - ±999,999.999	3
10000 - 99999	0.0001	0.0000 - ±99,999.9999	4
100000 - 999999	0.00001	0.00000 - ±9999.99999	5

NOTE

If scaling is desired for a particular path, SCALE must be enabled and all path scaling factors (PSCLA, PSCLD, PSCLV) must be specified prior to defining the path. SCALE cannot be enabled and scaling factors cannot be specified within a path definition.

Example: Refer to Define Path Local Mode (PL) command example.

PSCLV Path Velocity Scale Factor

		Product	Rev
Type	Scaling; Path Contouring or Motion (Linear Interpolated)	AT6n00	1.0
Syntax	<!>PSCLV<i>	AT6n50	1.0
Units	i = steps/unit	610n	n/a
Range	1 - 999,999	615n	n/a
Default	Steppers: 25000 Servos: Depends on feedback source for axis #1 (4000 if encoder, 819 if ANI, 432 if LDT)	620n	1.0
		625n	1.0
		6270	1.0
Response	PSCLV: *PSCLV25000		
See Also	PSCLA, PSCLD, PULSE, PV, SCALE, SFB		

The Path Velocity Scale Factor (PSCLV) command internally multiplies the path velocity (PV) value by this value. The PSCLV command will not scale the PV value unless the Scale command is enabled (SCALE1).

Steppers: The entered value is always in reference to motor steps, not encoder steps, regardless of the ENC command setting. If scaling is enabled (SCALE1), the entered velocity value is internally multiplied by the path velocity scaling factor (PSCLV) to convert user units/sec to motor steps/sec. If scaling is not enabled (SCALEØ), the PV value is entered in motor revs/sec; this value is internally multiplied by the drive resolution (DRES) value to obtain a velocity value in motor steps/sec for the motion trajectory calculations.

Servos: If scaling is enabled (SCALE1), the entered velocity value is internally multiplied by the path velocity scaling factor (PSCLV) to convert user units/sec to encoder, LDT or ANI steps/sec. If scaling is not enabled (SCALEØ), the PV value is entered in encoder revs/sec, LDT inches/sec, or ANI volts/sec; encoder and LDT values are internally multiplied by the encoder resolution (ERES) value or the LDT resolution (LDTRES) to obtain velocity values in steps/sec for the motion trajectory calculations.

As the path velocity scaling factor (PSCLV) changes, the resolution of the velocity commands and the number of positions to the right of the decimal point also change (see table below). A path velocity value with greater resolution than allowed will be truncated. For example, if scaling is set to PSCLV1Ø, the V1.9999 command would be truncated to V1.9.

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

Use the following equations to determine the maximum velocity range for your particular product:

Max. Velocity for Stepper Products		**Max. Velocity for Servo Products** (Servos: determined by feedback source selected for axis #1)	
$\frac{\left(\frac{8{,}000{,}000}{n}\right)}{\text{PSCLV}}$	*n* = PULSE x 16; If *n* < 5, then *n* is set equal to 5. If *n* > 5, then all fractional parts of *n* are truncated.	Encoder Feedback:	$\frac{1000 * \text{ERES}}{\text{PSCLV}}$
		LDT Feedback:	$\frac{1000 * \text{LDTRES}}{\text{PSCLV}}$
		ANI Feedback:	$\frac{1000 * 819}{\text{PSCLV}}$

NOTE

If scaling is desired for a particular path, scaling must be enabled (SCALE1) and all path scaling factors (PSCLA, PSCLD, PSCLV) must be specified prior to defining the path. Scaling cannot be enabled and scaling factors cannot be specified within a path definition.

Example: Refer to the define path local mode (PL) command example.

PSET — Establish Absolute Position

Type	Motion
Syntax	<!><@>PSET<r>,<r>,<r>,<r>
Units	r = units (absolute position)
Range	0.00000 - ±999,999,999.99999
Default	n/a
Response	n/a
See Also	ANIPOL, [CA], CMDDIR, D, ENC, ENCPOL, [FB], GO, HOM, INFNC, [LDT], LDTPOL, MA, MC, [PANI], [PC], [PCA], [PCC], [PCE], [PCL], [PCM], [PE], [PM], SCALE, SCLD, SFB, TCA, TFB, TLDT, TPANI, TPC, TPCA, TPCC, TPCE, TPCL, TPCM, TPE, TPM

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

Use the PSET command to offset the current absolute position to establish an *absolute position reference*. To remove the offset, issue the PSET CLR command (not available in earlier revisions of these products: 6200 rev 2.4, 6250 rev 3.0 and AT6400 rev 2.3).

All PSET values entered are in steps, unless scaling is enabled (SCALE1), in which case (PSET) is multiplied by the distance scale factor (SCLD):

Steppers – without scaling: In motor step mode (ENCØ), the PSET command will define the current motor step position to be the absolute position entered, but leave the encoder step position unchanged. In encoder step mode (ENC1), the PSET command will define the current encoder step position to be the absolute position given, but will leave the motor step position unchanged.

Servos – without scaling: The PSET command defines a new absolute position reference. If the drive is enabled (DRIVE1111), the current commanded position is used as the reference point. If the drive is disabled, the current feedback device position (selected with the SFB command) is used as the reference point.

NOTE

The PSET offset value (per axis) is specific only to the feedback source (per axis) selected with the last SFB command.

If your application requires switching between feedback sources for the same axis, then you must select the feedback source with the appropriate SFB command and issue a PSET value specific to that feedback source. (Each feedback source can have a separate offset.)

For 6270 users, the PSET settings for ANI and LDT feedback only are automatically saved to battery-backed RAM (encoder-based PSET is not saved).

Scaling: If scaling (SCALE) is enabled, the PSET command value entered is internally multiplied by the distance scaling factor (SCLD) to convert user units to motor steps, or feedback device (encoder, resolver, LDT, or ANI) steps. The distance value may be truncated if the value entered exceeds the distance resolution at the given scaling factor. The distance scaling factor should always be enabled and specified prior to entering the PSET value, because the SCLD command modifies the PSET value to accommodate the new scaling factor. For further discussion on distance scaling, refer to the SCLD command description.

NOTE: If you issue a PSET command, any previously captured positions (INFNCi-H function) will be offset by the PSET value.

If a software end-of-travel limit has been hit, the PSET command will not remove the error condition. The error condition is removed by commanding motion in the opposite direction.

Example:

```
PSET0,0,0,1000  ; Set absolute position on axes 1, 2, and 3 to zero,
                ; and axis 4 to 1000 units
```

[PSHF] Net Position Shift

		Product	Rev
Type	Following and Assignment or Comparison	AT6n00	3.0
Syntax	See below	AT6n50	3.0
Units	n/a	610n	4.0
Range	n/a	615n	3.0
Default	n/a	620n	3.0
Response	n/a	625n	3.0
See Also	ENC, FOLEN, FOLMAS, FSHFC, FSHFD, SCALE, SCLD, TPSHF	6270	3.0

The Net Position Shift (PSHF) command is used to assign to a numeric variable the value of the net (absolute) slave axis position shift that has occurred since that last FOLEN1 command. The position value will be the sum of all shifts performed on that axis, or axes, including decelerations due to limits, kill, or stop. The shift value is set to zero each time a new FOLEN1 command or a FOLMAS command (with a value other than zero) is issued.

Steppers: If scaling in enabled (SCALE1), the PSHF value is scaled by the distance scaling factor (SCLD). If scaling is not enabled, the value is in motor steps if in motor step mode (ENCØ) or in encoder steps if in encoder step mode (ENC1).

Servos: If scaling in enabled (SCALE1), the PSHF value is scaled by the distance scaling factor (SCLD). If scaling is not enabled, the value is commanded counts.

Syntax: VARn=aPSHF where n is the variable number and a is the axis number, or PSHF can be used in an expression such as IF(2PSHF>2345Ø). The PSHF command must be used with an axis specifier, or it will default to axis 1 (e.g., VAR1=1PSHF, IF(2PSHF>5ØØ), etc.).

Example:

```
IF(2PSHF>4.3)   ; If axis 2 has shifted more than 4.3 user units in the
                ; positive direction, then do the IF statement
OUT.12=b1       ; Set output #12 to 1
NIF             ; End of IF statement
VAR14=3PSHF     ; Set VAR14 to slave axis 3's position shift
```

[PSLV] Current Commanded Position of Slave

		Product	Rev
Type	Following and Assignment or Comparison	AT6n00	3.0
Syntax	See below	AT6n50	3.0
Units	n/a	610n	4.0
Range	n/a	615n	3.0
Default	n/a	620n	3.0
Response	n/a	625n	3.0
See Also	ENC, FMCNEW, FMCP, SCLD, SCALE, TPSLV	6270	3.0

The PSLV command is used to assign the slave's commanded position register value to a variable, or to make a comparison against another value.

Steppers: If scaling in enabled (SCALE1), the PSLV value is scaled by the distance scaling factor (SCLD). If scaling is not enabled, the value is in motor steps.

Servos: If scaling in enabled (SCALE1), the PSLV value is scaled by the distance scaling factor (SCLD). If scaling is not enabled, the value is commanded counts.

Syntax: VARn=aPSLV where n is the variable number and a is the axis number, or [PSLV] can be used in an expression such as IF(2PSLV>2345Ø). The PSLV command must be used with an axis specifier, or it will default to axis 1 (e.g., VAR1=1PSLV, IF(2PSLV>5ØØ), etc.).

Example:

```
IF(2PSLV>4.3)   ; If axis 2 has traveled more than 4.3 user units then do
                ; the IF statement
OUT.12=b1       ; Set output #12 to 1
NIF             ; End of IF statement
VAR14=3PSLV     ; Set VAR14 to slave axis #3's position
```

PTAN — Path Tangent Axis Resolution

Type	Path Contouring
Syntax	<!>PTAN<i>
Units	i = steps
Range	± 1 - 999,999,999
Default	25000
Response	No response - Must be defining a path (DEF)
See Also	PAXES

Product	Rev
AT6200	n/a
AT6400	1.0
AT6250	n/a
AT6450	4.1
610n	n/a
615n	n/a
620n	n/a
625n	n/a
6270	n/a

The Path Tangent Axis Resolution (PTAN) command is used to specify the Tangent axis resolution. The Tangent axis will keep an angular position which changes linearly with the direction of travel implied by X and Y. This allows the Tangent axis to control an object which must stay tangent (or normal) to the direction of travel.

The Tangent axis resolution is the number of motor steps in 360 degrees of arc. The Tangent axis resolution does not necessarily equal the number of steps per revolution of the motor, but if the motor directly drove the rotating piece, then these numbers would be the same.

The PTAN command should be given prior to any contour segments during a path definition. A negative value for the Tangent axis resolution causes rotation in the negative direction as the angle in the X-Y plane gets larger.

Example:

```
PSCLA25000      ; Set path acceleration scale factor to 25000 steps/unit/unit
PSCLD25000      ; Set path distance scale factor to 25000 steps/unit
PSCLV25000      ; Set path velocity scale factor to 25000 steps/unit
SCALE1          ; Enable scaling factor
PV5             ; Set path velocity to 5 units/sec
PA50            ; Set path acceleration to 50 units/sec/sec
PAD100          ; Set path deceleration to 100 units/sec/sec
DEF prog1       ; Begin definition of path named prog1
PAXES1,2,3      ; Set axes 1 and 2 as the X and Y contouring axes,
                ; 3 as the tangent axis
PTAN25000       ; Specify Tangent axis resolution
PAB0            ; Set to incremental coordinates
POUT1001        ; Output pattern during first arc
PARCM5,5,5      ; Specify incremental X-Y endpoint position and radius
                ; arc <180 degree for quarter circle counter-clockwise arc
POUT1100        ; Output pattern during second arc
PARCP5,-5,-5    ; Specify incremental X-Y endpoint position and radius
                ; arc >180 degree for three quarter circle clockwise arc
END             ; End definition of path prog1
PCOMP prog1     ; Compile path prog1
PRUN prog1      ; Execute path prog1
OUT0000         ; Turn off the first four programmable outputs
```

PUCOMP — Un-Compile a Compiled Profile (includes Path Uncompile)

Type	Compiled Motion; Path Contouring
Syntax	<!>PUCOMP<t>
Units	t = text (name of path)
Range	Text name of 6 characters or less
Default	n/a
Response	n/a
See Also	DEF, END, GOBUF, MEMORY, PCOMP, PRUN, TDIR, TMEM, TSEG, GOBUF, PLOOP, PLN

Product	Rev
AT6n00	1.0
AT6n50	4.1
610n	4.0
615n	4.1
620n	1.0
625n	4.1
6270	4.1

As of revision 4.0, contouring was made a standard feature in all multi-axis 6000 products (formerly only the steppers). In addition, PUCOMP was modified to accommodate uncompiling Compiled Motion Profiles.

The Un-Compile (PUCOMP) command is used to delete a previously compiled (PCOMP) program from the compiled memory. The PUCOMP command does not delete the program from program memory.

Example:
```
PUCOMP prog1    ; Delete compiled motion segments for prog1
DEL prog1       ; Delete prog1
DEF prog1       ; Begin definition of path named prog1
PAXES1,2,3,4    ; Set axes 1,2,3,4 as the X, Y, Tangent, and
                ; Proportional axes respectively
;  ******************************************
;  *  Add multiple path segment             *
;  *  definitions in this                   *
;  *  portion of the                        *
;  *  program                               *
;  ******************************************
END             ; End definition of path prog1
PCOMP prog1     ; Compile path prog1
PRUN prog1      ; Execute path prog1
```

PULSE Pulse Width

		Product	Rev
Type	Controller Configuration	AT6n00	1.0
Syntax	<!><@><a>PULSE<r>,<r>,<r>,<r>	AT6n50	n/a
Units	r = microseconds (μs)	610n	n/a
Range	0.3, 0.5, 1.0, 2.0, 5.0, 10.0, 16.0, or 20.0	615n	n/a
Default	0.3	620n	1.0
Response	PULSE: *PULSE0.3,0.3,0.3,0.3 1PULSE: *1PULSE0.3	625n	n/a
See Also	DRES, PCOMP, SCALE	6270	n/a

The Pulse Width (PULSE) command sets the step output pulse width. The pulse width is described as the time the pulse is active, or *on*. The value for the pulse width command is specified in microseconds.

When the pulse width is changed from the default value of 0.3 μs, the maximum velocity and distance ranges are reduced. The amount of reduction is directly proportional to the change in pulse width (see table below). The "maximum distance" is per move; the total absolute range for each axis remains at ±2,147,483,647.

Pulse Width (PULSE) Setting	Maximum Distance Per Move	Maximum Velocity
DEFAULT -- 0.3 μs	419,430,000	1.6 MHz
0.5 μs	262,140,000	1.0 MHz
Use for Compumotor's Z and DB Drives -- 1.0 μs	131,070,000	500 KHz
2.0 μs	65,535,000	250 KHz
5.0 μs	26,214,000	100 KHz
10.0 μs	13,107,000	50 KHz
16.0 μs	8,191,000	35 KHz
20.0 μs	6,553,000	25 KHz

NOTE

Contouring: All axes involved in contouring (identified with PAXES command) must have the same PULSE setting. If you change the PULSE setting, you will need to recompile (PCOMP) any previously compiled paths.

Streaming: All axes involved in the streaming mode (STREAM) must have the same PULSE setting.

Example:
```
PULSE2.0        ; Set the pulse width for axis 1 to 2.0 ms
```

PV — Path Velocity

Type	Path Contouring or Motion (Linear Interpolated)
Syntax	<!>PV<r>
Units	r = units/sec (scalable with PSCLV)
Range	0.00000 - 1,600,000 (depending on the scaling factor & PULSE)
Default	1.0000
Response	PV: *PV1.0000
See Also	GOL, PSCLV, SCALE

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	n/a
615n	n/a
620n	1.0
625n	1.0
6270	1.0

The Path Velocity (PV) command specifies the path velocity to be used in linearly interpolated moves (GOL), and in all contouring moves. In linearly interpolated moves, a path may involve one to four axes, each with its own distance of travel. In contouring paths, only the X and Y axis are included in the calculation of the path.

For both types of moves, the path velocity refers to the velocity of the load as motion proceeds along the path. For linearly interpolated moves, the velocity of each individual axis is dependent on the distance it contributes to the total path traveled by the load. In contouring paths, the velocity of each individual axis is dependent on the direction of travel in the X- Y plane. ***NOTE**: The PV value can be altered between path segments, but not within a path segment.*

Steppers: The entered value is always in reference to motor steps, not encoder steps, regardless of the ENC command setting. If scaling is not enabled (SCALEØ), the PV value is entered in motor revs/sec; this value is internally multiplied by the drive resolution (DRES) value to obtain a PV value in motor steps/sec for the motion trajectory calculations. If scaling is enabled (SCALE1), the entered PV value is internally multiplied by the path velocity scaling factor (PSCLV) to convert user units/sec to motor steps/sec.

Servos: If scaling is not enabled (SCALEØ), the PV value is entered in encoder revs/sec, LDT inches/sec, or ANI volts/sec; encoder and LDT values are internally multiplied by the encoder resolution (ERES) value or the LDT resolution (LDTRES) value to obtain a PV value in steps/sec for the motion trajectory calculations. If scaling is enabled (SCALE1), the entered PV value is internally multiplied by the path velocity scaling factor (PSCLV) to convert user units/sec to encoder, LDT, or ANI steps/sec.

Example: Refer to Define Path Local Mode (PL) command example.

PWC — Path Work Coordinates

Type	Path Contouring
Syntax	<!>PWC<r>,<r>
Units	r = units
Range	0.00000 - ±999,999,999
Default	0,0
Response	No response - Must be defining a path (DEF)
See Also	PAB, PL, PLC, PSCLD, SCALE

Product	Rev
AT6n00	1.0
AT6n50	4.1
610n	n/a
615n	n/a
620n	1.0
625n	4.1
6270	4.1

The Path Work Coordinates (PWC) command is used to specify the Work X -Y coordinate data required for subsequent segment definition in the Work coordinate system. This command places the X -Y coordinate value of the Work coordinate system at the beginning of the next segment. (The first <r> is the X coordinate, the second <r> is the Y coordinate.)

This command may be used before the PLØ command is given for the purpose of shifting the Work coordinate system. If the PWC command is not given before a PLØ command, but was previously set, the original work coordinate system is used for the subsequent segments.

Steppers only: Both position values are expressed in terms of motor steps, regardless of the current ENC command setting.

Scaling: If scaling (SCALE) is enabled, the PWC command values entered are internally multiplied by the path distance scaling factor (PSCLD) to convert user steps to motor steps. The distance values may be truncated if the value entered exceeds the distance resolution at the given scaling factor. For further discussion on path distance scaling, refer to the PSCLD command description.

Example: Refer to Define Path Local Mode (PL) command example.

RADIAN Radian Enable

		Product	Rev
Type	Operators (Trigonometric)		
Syntax	<!>RADIAN<b>	AT6n00	1.0
Units	n/a	AT6n50	1.0
Range	b = 0 (Disable), 1 (Enable) or X (don't care)	610n	4.0
Default	0	615n	1.0
Response	RADIAN: *RADIAN0	620n	1.0
See Also	[ATAN], [COS], [PI], [SIN], [TAN], VAR	625n	1.0
		6270	1.0

This operator is used to switch between radians and degrees. The command RADIAN1 specifies units in radians for SIN, COS, TAN, and ATAN. The command RADIANØ specifies units in degrees for SIN, COS, TAN, and ATAN.

If a value is given in radians and a conversion is needed to degrees, use the formula: 360° = 2π radians.

Example:

```
RADIAN1          ; Set trigonometric functions to radian mode
```

RE Registration Enable

		Product	Rev
Type	Registration		
Syntax	<!><@><a>RE<b><b><b><b>	AT6n00	1.4
Units	b = 0 (disable), 1 (enable), or X (don't care)	AT6n50	4.1
Range	n/a	610n	4.0
Default	0	615n	4.1
Response	RE: *RE0000	620n	1,5
	1RE: *1RE0	625n	4.1
See Also	[AS], COMEXC, ENC, [ER], INDEB, INFNC, [PCE], [PCM], REG, REGLOD, REGSS, [SS], TAS, TER, TPCE, TPCM	6270	4.1

The Registration Enable (RE) command enables the registration function for the appropriate axes.

When a registration input (assigned trigger input) is activated, the motion profile currently being executed is replaced by the registration profile with its own distance (REG), acceleration (A & AA), deceleration (AD & ADA), and velocity (V) values, and, if using a stepper, the positioning mode (ENC). The registration move may interrupt any preset, continuous, or registration move in progress.

The registration move does not alter the rest of the program being executed when registration occurs, nor does it affect commands being executed in the background if the controller is operating in the continuous command execution mode (COMEXC1).

Registration moves will not be executed while the motor is not performing a move, while in the joystick mode (JOY1), or while decelerating due to a stop, kill, soft limit, or hard limit.

How to Set up a Registration Move

- Configure one of the trigger inputs (TRG-A through TRG-D) to function as a trigger interrupt input; this is done with the INFNCi-H command, where i is the input bit number representing trigger inputs A through D (input number assignments vary by product – see page 6). NOTE: When configured as Trigger Interrupts, the triggers cannot be affected by the input enable (INEN) command.

How to Set up a Registration Move *(continued)*

- Issue the INFEN1 command to enable the trigger interrupt/registration function defined with the INFNCi-H command.
- Specify the distance of the registration move with the REG command; then you can enable the registration function with the RE command. Registration is performed only on the axis or axes with the registration function enabled, and with a non-zero distance specified in the respective axis-designation field of the REG command; the other axes will not be affected. Each trigger has a distinct move defined for each axis; therefore, with 4 trigger inputs and 4 axes available, 16 different registration moves can be stored.

  NOTE: The registration move is executed using the A, AA, AD, ADA, and V values that were in effect when the REG command was entered. Stepper products: The position captured (motor or encoder) and the positioning mode (motor steps or encoder steps) used for registration are determined by the ENC command setting in effect when the registration move was first defined with the REG command.

Registration Move Accuracy (see also Registration Move Status below)

The registration move distance (specified with the REG command) is based on the actual position captured when the registration input is activated. Therefore, the accuracy of the registration move is determined by the slight lapse between activating the registration input and capturing the position. The accuracy is calculated as 50μs * the velocity of the axis at the time the input was activated. The exception to this rule is if you are using a servo product's dedicated hardware latch triggers (triggers A-D are dedicated to encoders 1-4, respectively), in which case the registration move accuracy is ±1 encoder count.

RULE OF THUMB: To prevent position overshoot, make sure the REG distance is greater than 4 ms multiplied by the incoming velocity.

The lapse between activating the registration input and commencing the registration move (this does not affect the move accuracy) is less than one position sample period. The sample period for steppers is 2 ms. The sample period for servos is determined by the SSFR and INDAX command settings (refer to the System Update column in the table in the SSFR command description to ascertain your controller's sample period).

The REG distance will be scaled by the distance scale factor (SCLD value) if scaling is enabled (SCALE1).

Preventing Unwanted Registration Moves (methods)

- Registration Input Debounce: The registration input is debounced for 50 ms (steppers) or 24 ms (servos) before another input on the same trigger is recognized. Therefore, the maximum rate that a registration input can initiate registration moves is 20 times per second (steppers) or 41 times per second (servos).

  If your application requires a shorter debounce time, you can change it with the INDEB command (refer to the INDEB command description for details).
- Registration Single-Shot: The REGSS command allows you to program the 6000 controller to ignore any registration commands after the first registration move has been initiated. Refer to the REGSS command description for further details and an application example.
- Registration Lockout Distance: The REGLOD command specifies what distance an axis must travel before any trigger assigned as a registration input will be recognized. If more than one axis is using the same trigger for registration, and one or all have a registration lockout distance defined, then that trigger will be ignored until all axes have traveled the lockout distances. Refer to the REGLOG command description for further details and an application example.

Registration Move Status & Error Handling

Axis Status — Bit #28: This status bit is set when a registration move has been initiated by any registration input (trigger). This status bit is cleared with the next GO command.

AS.28.....Assignment & comparison operator — use in a conditional expression.
TASF.....Full text description of each status bit. (see "Reg Move Commanded" line item)
TAS.....Binary report of each status bit (bits 1-32 from left to right). See bit #28.

Axis Status — Bit #30: If, when the registration input is activated, the registration move profile cannot be performed with the specified motion parameters, the 6000 controller will kill the move in progress and set axis status bit #30. This status bit is cleared with the next GO command.

AS.30.....Assignment & comparison operator — use in a conditional expression.
TASF.....Full text description of each status bit. (see "Preset Move Overshot" line item)
TAS.....Binary report of each status bit (bits 1-32 from left to right). See bit #30.

Error Status — Bit #10: This status bit may be set if axis status bit #30 is set. The error status is monitored and reported only if you enable error-checking bit #10 with the ERROR command (e.g., ERROR.10-1). NOTE: When the error occurs, the controller with branch to the error program (assigned with the ERRORP command). This status bit is cleared with the next GO command.

ER.10.....Assignment & comparison operator — use in a conditional expression.
TERF.....Full text description of each status bit. (see "Preset Move Overshot" line item)
TER.....Binary report of each status bit (bits 1-32 from left to right). See bit #10.

System Status — Bits #25-28: System status bits 25-28 are set when a registration move has been initiated by trigger inputs A through D, respectively. This also indicates that the positions of all axes has been captured. As soon as the captured information is transferred or assigned/compared, the respective system status bit is cleared (set to 0).

SS.....Assignment & comparison operator — use in a conditional expression. Operators for bits 25-28 are SS.25, SS.26, SS.27, and SS.28.
TSSF.....Full text description of each status bit. (see "Position Captured TRG- n")
TSS.....Binary report of each status bit (bits 1-32 from left to right). See bits #25-28.

Example:

In this example, two-tiered registration is achieved. While axes 1 & 2 are executing their 200,000-unit moves, trigger input A is activated and executes registration move A to slow the load's movement. An open container of volatile liquid is then placed on the load. After picking up the liquid and while registration move A is still in progress, trigger input B is activated and executes registration move B to gently slow the load to an even lower velocity before motion is gently stopped.

```
INFNC25-H       ; Set input #25 (trigger A) as a trigger interrupt input
INFNC26-H       ; Set input #26 (trigger B) as a trigger interrupt input
INFEN1          ; Enable programmable input (INFNC) functions
ENC00xx         ; Use motor step positioning mode for registration
                ; move (STEPPERS ONLY)
A10,20          ; Set accel: axis 1 to 10 units/sec/sec;
                ; axis 2 to 20 units/sec/sec
AD20,40         ; Set decel: axis 1 to 20 units/sec/sec;
                ; axis 2 to 40 units/sec/sec
V2,5            ; Set velocity: axis 1 to 2 units/sec; axis 2 to 5 units/sec
REGA1000,5000   ; Set trigger A's registration distance on axis 1 to 1000
                ; units, axis 2 to 5000 units (registration A move will use
                ; the ENC, A, AD, & V values specified above)
ENC00xx         ; Use the motor step positioning mode for the registration
                ; move (STEPPERS ONLY)
A3,5            ; Set accel: axis 1 to 3 units/sec/sec; axis 2 to 5 units/sec/sec
AD2,4           ; Set decel: axis 1 to 2 units/sec/sec; axis 2 to 4 units/sec/sec
V.5,.5          ; Set velocity: axis 1 and axis 2 to 0.5 units/sec
REGB800,1200    ; Set trigger B's registration distance on axis 1 to 800 units,
                ; axis 2 to 1200 units (registration B move will use the ENC,
                ; A, AD, & V values specified above)
RE1100          ; Enable registration on axis 1 and 2 only
A20,20          ; Set acceleration to 20 units/sec/sec on axes 1 and 2
AD50,50         ; Set deceleration to 50 units/sec/sec on axes 1 and 2
V6,6            ; Set velocity to 6 units/sec on axes 1 and 2
D200000,200000  ; Set distance to 200,000 units on axes 1 and 2
GO1100          ; Initiate motion on axes 1 and 2
```

[READ] Read a Value

		Product	Rev
Type	Communication Interface or Assignment	AT6n00	1.0
Syntax	... READi ... (See below)	AT6n50	1.0
Units	i = string variable number	610n	4.0
Range	1 - 100 (AT6400), 1 - 25 (AT6n50, 610n, 615n, 620n, 625n, and 6270)	615n	1.0
Default	n/a	620n	1.0
Response	n/a	625n	1.0
See Also	', PORT, [SS], TSS, VAR, VARS, WRITE	6270	1.0

The Read a Value (READ) command provides the user with an efficient way of storing numeric data read from the input buffer into a variable (PC-AT would place data in the input buffer, data from stand-alone units comes from RS-232). The READ command can be used as part of a numeric variable assignment statement (e.g., VAR1=READ1) or in another command (A1Ø,(READ1),12,1). However, the READ command cannot be used in an expression such as VAR5=1+READ1 or IF(READ1=1).

Syntax: VARx=READi where x is the variable number and i is the string variable to be sent out to prompt the user for the numeric information.

Syntax: Command(READi) where Command is any command that has a separate field (e.g., A, AD, V, D, etc.), and i is the string variable number.

The number attached to the end of the READ command corresponds to the string variable to be placed in the PC-AT output buffer, or sent out the RS-232 port, at the time this command is executed. The 6000 Series controller will then wait for numeric data to be sent to its input buffer. **The numeric data must be preceded with an immediate command identifier and a single quote (!').** The information read in can be either integer, or real, and must be terminated by a command delimiter (:, <cr>, <lf>).

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

Example:

```
VARS1="Enter the count >" ; Place message in string variable #1
VAR2=READ1                ; Prompt with string variable #1, and read data
                          ; into variable #2
;
; The controller will send this message (string variable #1) to the screen:
;                  "Enter the count >"
; The user must enter the numeric data preceded by the characters !'.
; For example, !'82.5 assigns the value 82.5 to numeric variable 2
```

REG Registration Distance

		Product	Rev
Type	Registration	AT6n00	1.4
Syntax	<!><@><a>REGc<r>,<r>,<r>,<r>	AT6n50	4.1
Units	c = letter of trigger input; r = distance units (scalable)	610n	4.0
Range	c = A, B, C or D; r = 0.00000 to 419,430,000 (positive direction only)	615n	4.1
Default	0 (do not make a registration move)	620n	1.5
Response	REGA: *REGA0,0,0,0 1REGA: *1REGA0	625n	4.1
See Also	[AS], ENC, [ER], INDEB, [PCE],[PCM], RE, REGLOD, REGSS, SCALE, SCLD, [SS], TAS, TER, TPCE, TPCM, TSS	6270	4.1

The Registration Distance (REG) command specifies the distance the corresponding axis will travel after receiving a registration input (trigger A, B, C or D).

The registration move is executed using the A, AA, AD, ADA, and V values that were in effect when the REG command was entered. Stepper products: The position captured (motor or encoder) and the positioning mode (motor steps or encoder steps) used for registration are determined by the ENC command setting in effect when the registration move was first defined with the REG command.

RULE OF THUMB: To prevent position overshoot, make sure the REG distance is greater than 4 ms multiplied by the incoming velocity.

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

Distance Units (no scaling):

In stepper systems with scaling disabled (SCALEØ), the REG value is measured in encoder counts or motor steps, depending on the state of the ENC command at the time the REG command is executed.

In servos systems, the REG value is measured in feedback device counts (encoder, resolver, LDT, or ANI counts).

Scaling: If scaling is enabled (SCALE1), the REG value is internally multiplied by the distance scale factor (SCLD). As the SCLD value changes, the resolution of the REG command and the number of positions to the right of the decimal point also change. A registration distance value with greater resolution than allowed will be truncated. For example, if scaling is set to SCLD2ØØØØ, the REG1.99999 command would be truncated to REG1.9999.

SCLD **(steps/unit)**	REG **Resolution (units)**	REG **Range (units)**	**Decimal Places**
1 - 9	1	0 - 999,999,999	0
10 - 99	0.1	0.0 - 99,999,999.9	1
100 - 999	0.01	0.00 - 9,999,999.99	2
1000 - 9999	0.001	0.000 - 999,999.999	3
10000 - 99999	0.0001	0.0000 - 99,999.9999	4
100000 - 999999	0.00001	0.00000 - 9,999.99999	5

The distance scaling factor should always be enabled and specified prior to entering any distance values, because the SCLD command modifies the current registration distance value to accommodate the new scaling factor.

For additional details on Registration (including programming examples), refer to the RE command description and to the Registration section in the *6000 Series Programmer's Guide*.

REGLOD Registration Lock-Out Distance

Type	Registration
Syntax	<!>@REGLOD<r>,<r>,<r>,<r>
Units	r= distance units (scalable by SCLD)
Range	0.00000 to +999,999,999
Default	0
Response	REGLOD: *REGLOD0,0,0,0 1REGLOD: *1REGLOD0
See Also	INDEB, RE, REG, REGSS, SCLD

Product	**Rev**
AT6n00	4.1
AT6n50	4.1
610n	4.0
615n	4.1
620n	4.1
625n	4.1
6270	4.1

The Registration Lock-Out Distance (REGLOD) command sets the registration lock-out distance for an axis. The controller will not perform a registration move or a position capture on any axis until all axes have traveled the distances specified with the REGLOD command from their respective positions when motion was initiated. Those axes participating in registration with trigger A, B, C, or D are defined with the REG command (REGA, REGB, REGC, or REGD, respectively). A trigger will not be recognized until all axes participating in that trigger's registration have traveled their respective lock-out distances.

If scaling is enabled (SCALE1), the lock-out distance is scaled by the SCLD value.

Stepper products: the lock-out distances are measured incrementally from the start of motion to the commanded position (even if you are using encoder-based positioning in the ENC1 mode).

Servo products: the lock-out distances are measured incrementally from the start of motion to the actual position (as measured by the position feedback device), not the commanded position.

Example:

```
INFNC25-h       ; Set trigger interrupt
REGA25000,25000 ; Set registration distance
RE11            ; Enable registration axis 1 and 2
REGLOD20000     ; Set registration lock-out distance for axis 1
@D50000         ; Set move distance for both axes
GO11            ; Start motion for axes 1 and 2
```

If the trigger happens when axis 1 is at any distance less than 20000, no motion will occur on any axis.

```
> !TAS.28       Check status bit 28 for "registration occurred" bit
  *00           Indicates registration move has not happened on any axis
```

If the trigger happens after axis 1 has traveled a distance of 20000, then the trigger is recognized and the registration occurs.

REGSS Registration Single-Shot

Type	Registration
Syntax	<!><@>REGSS<b><b><b><b>
Units	n/a
Range	b=0 (Disable), 1 (Enable), or x (don't care)
Default	0
Response	REGSS: *REGSS0000 1REGSS: *1REGSS0
See Also	RE, REG, REGLOD

Product	Rev
AT6n00	4.1
AT6n50	4.1
610n	4.0
615n	4.1
620n	4.1
625n	4.1
6270	4.1

The Registration Single Shot (REGSS) command sets the registration such that only one registration move will take place for the specified axis. This allows the user to prevent any other trigger from interrupting the registration move in progress. A GO command will reset the "one shot" condition.

Example:

```
INFNC25-h       ; Set trigger interrupt
REGA25000,25000 ; Set registration distance
RE11            ; Enable registration axis 1 and 2
REGSS1          ; Enable single-shot registration for axis 1
@D50000         ; Set move distance for both axes
GO11            ; Start motion for axes 1 and 2
```

The trigger happens at distance 30000, both axes start registration move.

```
> !TAS.28                 Check status bit 28 for "registration occurred" bit
  *11                     Indicates registration move happened on both axes
```

The trigger happens at distance 50000, axis 2 starts the second registration move, axis 1 is not affected.

```
> TPM                     Check final motor position when registration moves stop — steppers
                          (or, if using a servo product, use TPC to check commanded position)
  *TPM+55000,+75000       Only axis 2 responds to the second trigger
```

REPEAT Repeat Statement

		Product	Rev
Type	Program Flow Control or Conditional Branching	AT6n00	1.0
Syntax	<!>REPEAT	AT6n50	1.0
Units	n/a	610n	4.0
Range	n/a	615n	1.0
Default	n/a	620n	1.0
Response	n/a	625n	1.0
See Also	JUMP, UNTIL	6270	1.0

The Repeat Statement (REPEAT) command, in conjunction with the UNTIL command, provide a means of conditional program flow. The REPEAT command marks the beginning of the conditional statement. The commands between the REPEAT and the UNTIL command are executed at least once. Upon reaching the UNTIL command, the expression contained within the UNTIL command is evaluated. If the expression is false, the program flow is redirected to the first command after the REPEAT command. If the expression is true, the first command after the UNTIL command is executed.

Up to 16 levels of REPEAT ... UNTIL( ) commands may be nested.

NOTE: Be careful about performing a GOTO between REPEAT and UNTIL. Branching to a different location within the same program will cause the next REPEAT statement encountered to be nested within the previous REPEAT statement, unless an UNTIL command has already been encountered. The JUMP command should be used in this case.

All logical operators (AND, OR, NOT), and all relational operators (=, >, >=, <, <=, <>) can be used within the UNTIL expression. There is no limit on the number of logical operators, or on the number of relational operators allowed within a single UNTIL expression.

The limiting factor for the UNTIL expression is the command length. The total character count for the UNTIL command and expression cannot exceed 80 characters. For example, if you add all the letters in the UNTIL command and the letters within the ( ) expression, including the parentheses and excluding the spaces, this count must be less than or equal to 80.

All assignment operators (A, AD, ANV, AS, CNT, D, ER, IN, INO, LIM, MOV, OUT, PC, PCE, PCM, PE, PER, PM, SS, TIM, US, V, VEL, etc.) can be used within the UNTIL( ) expression.

Example:

```
REPEAT          ; Beginning of REPEAT ... UNTIL( ) loop
GO1110          ; Initiate motion on axes 1, 2, and 3
VAR1=VAR1+1     ; Increment variable 1 by 1
UNTIL(VAR1=12)  ; Repeat loop until variable 1 = 12
```

RESET — Reset

Type	Communication Interface
Syntax	<!>RESET
Units	n/a
Range	n/a
Default	n/a
Response	RESET: *Parker Compumotor AT6400 - 4 Axis Indexer
See Also	DRESET, STARTP, TSTAT

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Reset (RESET) command functions differently, depending on which 6000 Series product you have.

- The RESET command returns bus-based products to their factory default state. All previously entered command values will be reset to factory default. All programs/subroutines will be removed.
- The RESET command affects stand-alone products the same as cycling power. These products will retain their programs and variables; however, all previously entered command values (not saved in programs or variables) will be reset to factory default.

NOTE: After sending the RESET or !RESET command to the 6000 product, you must wait until you see the power-up message (actual time varies by product) before communicating with the product.

RUN — Begin Executing a Program

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

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Begin Executing a Program (RUN) command executes a program defined with the DEF command. A program name consists of 6 or fewer alpha-numeric characters. The RUN command can be used inside a program or subroutine. The program can also be run by specifying the name of the program without the RUN command. The RUN command functions similar to a GOSUB command in that control returns to the original program when the called program finishes.

Example:

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

S — Stop Motion

		Product	Rev
Type	Motion	AT6n00	1.0
Syntax	<!>S<b><b><b><b>	AT6n50	1.0
Units	n/a	610n	4.0
Range	b = 0 (do not stop) or 1 (stop)	615n	1.0
Default	1	620n	1.0
Response	>!S: No response, instead motion is stopped on all axes.	625n	1.0
See Also	C, COMEXC, COMEXS, GO, K	6270	1.0

The Stop Motion (S) command instructs the motor to stop motion on the specified axes. If the Stop (S) command is used without any arguments, motion will be stopped on all axes. The Stop command will bring the specified axes to rest using the last deceleration value (AD) entered.

NOTE

Since all commands are buffered, the next command does not begin until the previous command has finished. This is important because if you place a Stop (S) command after a Go (GO) command in a program, the Stop command will have no effect. For the Stop command to have an effect within a program, continuous command processing mode (COMEXC) must be enabled. If the Stop (S) command is to be used external to the program, the immediate command identifier (!) must be used.

If COMEXS is set to zero, command processing will be terminated when any stop command is issued, or a stop input is activated. If COMEXS is set to 1 or 2, a stop command issued for a specific axis will only stop motion on that axis and will not clear the command buffer. If COMEXS is set to 2, a stop command or input will stop motion and clear the command buffer.

If motion is to be paused and later resumed, the stop command must be used without any arguments (S or !S), and the continue execution on stop (COMEXS) command must be enabled. The continue (!C) command can then be used to resume motion.

Example:

```
GO1111   ; Initiate motion on all axes
!S1100   ; Stop motion on axes 1 and 2 (must use "!S" to stop motion in progress)
```

SCALE — Enable/Disable Scale Factors

		Product	Rev
Type	Scaling	AT6n00	1.0
Syntax	<!>SCALE<b>	AT6n50	1.0
Units	n/a	610n	4.0
Range	b = 0 (disable) or 1 (enable)	615n	1.0
Default	0 (1 for 6270 only)	620n	1.0
Response	SCALE: *SCALE0	625n	1.0
See Also	DRES, ERES, PSCLA, PSCLD, PSCLV, SCLA, SCLD, SCLMAS, SCLV, SFB, TSTAT	6270	1.0

The Enable Scale Factor (SCALE) command enables or disables the acceleration, distance, and velocity scaling factors (SCLA, SCLD, SCLV, PSCLA, PSCLD, PSCLV, SCLMAS). When scaling is enabled (SCALE1), all entered data is multiplied by the appropriate scale factor.

Servos: Scaling can be used with all feedback sources: encoders, resolvers, ANI inputs (-ANI option only), and LDTs (6270 only). Accel and decel values are scaled by SCLA and PSCLA, velocity values are scaled by SCLV and PSCLV, and distance values are scaled by SCLD.

NOTE

Parameters for scaling (SCLA, SCLD, etc.) are specific to the feedback source selected with the last SFB command. Therefore, if your application requires switching between feedback sources for the same axis, then for each feedback source, you must select the feedback source with the appropriate SFB command and issue the scaling factors specific to that feedback source.

When scaling is disabled (SCALEØ), no scaling will be performed:

- **Steppers**: When scaling is disabled, all distance values entered are in motor steps (ENCØ mode) or encoder steps (ENC1 mode), and all acceleration and velocity values entered are internally multiplied by the DRES command value.
- **Servos**: When scaling is disabled, all distance values entered are in encoder, resolver or LDT counts, or ADC counts for ANI feedback. All encoder, resolver and LDT accel/decel and velocity values entered are internally multiplied by the ERES or LDTRES command value (ANI accel/decel and velocity values are referenced in volts).

NOTE

If scaling is desired within a stored program, you must enable scaling (SCALE1) and define the scaling factors (SCLA, SCLD, SCLV, PSCLA, PSCLD, PSCLV, SCLMAS) prior to defining (DEF), uploading (TPROG), or running (RUN) the program. This allows the 6000 Series product to store, display, and execute the scaled distance, acceleration, and velocity values within the stored program. This can be accomplished by defining all scaling factors via a terminal emulator just before defining or downloading the program; or you can put the scaling factors into a startup (STARTP) program (stand-alone controllers only) or a program that must be run prior to defining or downloading the program.

Example: Refer to the programming example for the Acceleration Scale Factor (SCLA) command.

SCLA Acceleration Scale Factor

Type	Scaling
Syntax	<!><@><a>SCLA<i>,<i>,<i>,<i>
Units	i = steps/unit
Range	1 - 999,999
Default	Steppers: 25000 Servos: Depends on feedback source(4000 if encoder, 4096 if resolver, 819 if ANI, 432 if LDT)
Response	SCLA: *SCLA25000,25000,25000,25000 1SCLA: *1SCLA25000
See Also	FMAXA, SCALE, SCLD, SCLV, SFB, TSTAT

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Acceleration Scale Factor (SCLA) command internally multiplies all acceleration (A, AA, HOMA, HOMAA, JOGA, JOGAA, JOYA, JOYAA) and deceleration (AD, ADA, LHAD, LHADA, LSAD, LSADA, HOMAD, HOMADA, JOGAD, JOGADA, JOYAD, JOYADA) values by the acceleration scale factor value, as long as scaling is enabled (SCALE1). Since the units are steps/unit, and all the acceleration values are in units/sec/sec, all accelerations will thus be internally represented as steps/sec/sec.

Stepper products: If scaling is enabled (SCALE1), the entered accel and decel values are internally multiplied by the acceleration scaling factor (SCLA) to convert user units/sec/sec to motor steps/sec/sec. The entered values are always in reference to motor steps, not encoder steps, regardless of the ENC command setting.

If scaling is disabled (SCALEØ), all accel and decel values are entered in motor revs/sec/sec; these values are internally multiplied by the drive resolution (DRES) value to obtain accel and decel values in motor steps/sec/sec for the motion trajectory calculations.

Servo products: If scaling is enabled (SCALE1), the entered accel and decel values are internally multiplied by the acceleration scaling factor (SCLA) to convert user units/sec/sec to encoder, resolver, LDT, or ANI steps/sec/sec. If scaling is disabled (SCALEØ), all accel and decel values are entered in encoder revs/sec/sec, resolver revs/sec/sec, LDT inches/sec/sec, or ANI volts/sec/sec; encoder, resolver and LDT values are internally multiplied by the encoder resolution (ERES) value or the LDT resolution (LDTRES) value to obtain accel and decel values in steps/sec/sec for the motion trajectory calculations.

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

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

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

Product	**Min. Accel or Decel (resolution)**	**Max. Accel or Decel**
Steppers	$\frac{0.001 * \text{DRES}}{\text{SCLA}}$	$\frac{999.9999 * \text{DRES}}{\text{PSCLA}}$
Servos	Encoder & Resolver Feedback: $\frac{0.001 * \text{ERES}}{\text{SCLA}}$	Encoder & Resolver Feedback: $\frac{999.9999 * \text{ERES}}{\text{SCLA}}$
	LDT Feedback: $\frac{0.001 * \text{LDTRES}}{\text{SCLA}}$	LDT Feedback: $\frac{999.9999 * \text{LDTRES}}{\text{SCLA}}$
	ANI Feedback: $\frac{0.819}{\text{SCLA}}$	ANI Feedback: $\frac{818999.9181}{\text{SCLA}}$

NOTE

If scaling is desired within a stored program, you must enable scaling (SCALE1) and define the scaling factors (SCLD, SCLA, and SCLV) prior to defining (DEF), uploading (TPROG), or running (RUN) the program. This allows the 6000 Series product to store, display, and execute the scaled distance, acceleration, and velocity values within the stored program. This can be accomplished by defining all scaling factors via a terminal emulator just before defining or downloading the program; or you can put the scaling factors into a startup (STARTP) program (stand-alone controllers only) or a program that must be run prior to defining or downloading the program.

Example:

```
SCLA25000,25000,1,1 ; Set the acceleration scaling factor for axes 1 and 2 to
                    ; 25000 steps/unit/unit, axes 3 and 4 to 1 step/unit/unit
SCLV25000,25000,1,1 ; Set the velocity scaling factor for axes 1 and 2 to 25000
                    ; steps/unit, axes 3 and 4 to 1 step/unit
SCLD1,1,1,1         ; Set the distance scaling factor for axes 1, 2, 3, and 4 to
                    ; 1 step/unit
SCALE1              ; Enable scaling
DEF prog1           ; Begin definition of program prog1
MA0000              ; Incremental index mode for all axes
MC0000              ; Preset index mode for all axes
A10,12,1,2          ; Set the acceleration to 10, 12, 1, and 2 units/sec/sec
                    ; for axes 1, 2, 3 and 4
V1,1,1,2            ; Set the velocity to 1, 1, 1, and 2 units/sec
                    ; for axes 1, 2, 3 and 4
D100000,1000,10,100 ; Set the distance to 100000, 1000, 10, and 100 units
                    ; for axes 1, 2, 3 and 4
GO1100              ; Initiate motion on axes 1 and 2, 3 and 4 do not move
END                 ; End definition of program prog1
```

SCLD Distance Scale Factor

Type	Scaling
Syntax	<!><@><a>SCLD<i>,<i>,<i>,<i>
Units	i = steps/unit
Range	1 - 999,999
Default	Steppers: 1 Servos: Depends on feedback source (1 if encoder or resolver, 819 if ANI, 432 if LDT)
Response	SCLD: *SCLD1,1,1,1 1SCLD: *1SCLD1
See Also	[ANI], [CA], D, [FB], FOLRN, FSHFD, [LDT], [PANI], [PC], [PCA], [PCC], [PCE], [PCL], [PCM], [PE], [PER], [PM], PSET, [PSHF], [PSLV], REG, REGLOD, SCALE, SCLA, SCLV, SCLMAS, SFB, SMPER, TANI, TCA, TFB, TLDT, TPANI, TPC, TPCA, TPCC, TPCE, TPCL, TPCM, TPE, TPER, TPM, TPSHF, TPSLV, TSTAT

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

If scaling is enabled (SCALE1), all D, PSET, SMPER, and REG command values are internally multiplied by the Distance Scale Factor (SCLD) value. Since the SCLD units are in terms of steps/unit, all distances will thus be internally represented in steps. For instance, if your distance scaling factor is 10000 (SCLD1ØØØØ) and you enter a distance of 75 (D75), the actual distance moved will be 750,000 (10000 x 75) counts.

This command is useful for allowing the user to specify distances in any unit. For example, if the user had a 25000 step/revolution drive and wanted distance units in terms of revolutions, then SCLD should be set to 25000, and scaling should be enabled (SCALE1).

As the distance scaling factor (SCLD) changes, the resolution of all distance commands and the number of positions to the right of the decimal point also change (see table below). A distance value with greater resolution than allowed will be truncated (e.g., if scaling is set to SCLD25ØØØ, the D1.99999 command would be truncated to D1.9999). For 6270 users only, shift the decimal place in the distance ranges shown in the table below one place to the left.

SCLD (steps/unit)	Distance Resolution (units)	Distance Range (units)	Decimal Places
1 - 9	1	0 - ±999,999,999	0
10 - 99	0.1	0.0 - ±99,999,999.9	1
100 - 999	0.01	0.00 - ±9,999,999.99	2
1000 - 9999	0.001	0.000 - ±999,999.999	3
10000 - 99999	0.0001	0.0000 - ±99,999.9999	4
100000 - 999999	0.00001	0.00000 - ±9999.99999	5

DEFINE SCALING FIRST

If scaling is desired within a stored program, you must enable scaling (SCALE1) and define the scaling factors (SCLD, SCLA, and SCLV) prior to defining (DEF), uploading (TPROG), or running (RUN) the program. This allows the 6000 Series product to store, display, and execute the scaled distance, acceleration, and velocity values within the stored program. This can be accomplished by defining all scaling factors via a terminal emulator just before defining or downloading the program; or you can put the scaling factors into a startup (STARTP) program (stand-alone controllers only) or a program that must be run prior to defining or downloading the program.

FRACTIONAL STEP TRUNCATION

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

Example:

```
SCLA25000,25000,1,1 ; Set the acceleration scaling factor for axes 1 and 2 to
                    ; 25000 steps/unit/unit, axes 3 and 4 to 1 step/unit/unit
SCLV25000,25000,1,1 ; Set velocity scaling factor for axes 1 and 2 to 25000
                    ; steps/unit, axes 3 and 4 to 1 step/unit
SCLD1,1,1,1         ; Set distance scaling for axes 1, 2, 3, & 4 to 1 step/unit
SCALE1              ; Enable scaling
DEF prog1           ; Begin definition of program prog1
MA0000              ; Incremental index mode for all axes
MC0000              ; Preset index mode for all axes
A10,12,1,2          ; Set accel to 10, 12, 1, and 2 units/sec/sec for axes 1-4
V1,1,1,2            ; Set velocity to 1, 1, 1, and 2 units/sec for axes 1-4
D100000,1000,10,100 ; Set distance to 100000, 1000, 10, and 100 units for axes 1-4
GO1100              ; Initiate motion on axes 1 and 2, 3 and 4 do not move
END                 ; End definition of program prog1
```

SCLMAS Master Scale Factor

Type	Following
Syntax	<!><@><a>SCLMAS<i>,<i>,<i>,<i>
Units	i = scaling factor
Range	i = 1 - 999999
Default	1
Response	SCLMAS *SCLMAS1,1,1,1 1SCLMAS *1SCLMAS1
See Also	FMCLEN, FMCP, FOLEN, FOLMD, FOLRD, FOLRN, GOWHEN, [PMAS], SCALE, SCLD, TPMAS

Product	Rev
AT6n00	3.0
AT6n50	3.0
610n	4.0
615n	3.0
620n	3.0
625n	3.0
6270	3.0

The Master Scale Factor (SCLMAS) command internally multiplies all Following master values by the specified scale factor value. Since the SCLMAS units are in terms of counts/unit, all distances will thus be internally represented in counts. For instance, if your master scaling factor is 10000 (SCLMAS10000) and you enter a master parameter of 75 (e.g., FOLMD75), the internal value will be 750,000 (10000 x 75) counts.

NOTE: The SCLMAS command will not take effect unless scaling is enabled (SCALE1).

This command is useful for allowing the user to specify distances in any unit. For example, if the user had a 4000 step/revolution encoder as the master and wanted master units in terms of revolutions, then SCLMAS should be set to 4000.

As the master scaling factor (SCLMAS) changes, the resolution of all master parameter values and the number of positions to the right of the decimal point also change (see table below). A master parameter value with greater resolution than allowed will be truncated (e.g., if scaling is set to SCLD4000, the FOLMD1.9999 command would be truncated to FOLMD1.999). (For 6270 users, shift the values in the "Master Range" column one decimal place to the left).

SCLMAS (counts/unit)	Master Resolution (units)	Master Range (units)	Decimal Places
1 - 9	1	0 - ±999,999,999	0
10 - 99	0.1	0.0 - ±99,999,999.9	1
100 - 999	0.01	0.00 - ±9,999,999.99	2
1000 - 9999	0.001	0.000 - ±999,999.999	3
10000 - 99999	0.0001	0.0000 - ±99,999.9999	4
100000 - 999999	0.00001	0.00000 - ±9999.99999	5

DEFINE SCALING FIRST

If scaling is desired within a stored program, you must enable scaling (SCALE) and define the scaling factor (SCLMAS) prior to defining (DEF), uploading (TPROG), or running (RUN) the program. This allows the 6000 Series product to store, display, and execute the scaled distance, acceleration, and velocity values within the stored program. This can be accomplished by defining all scaling factors via a terminal emulator just before defining or downloading the program; or you can put the scaling factors into a startup (STARTP) program (stand-alone controllers only) or a program that must be run prior to defining or downloading the program.

FRACTIONAL STEP TRUNCATION

If you are specifying master distance values (FOLMD), when the master scaling factor (SCLMAS) and the distance value are multiplied, a fraction of one count may possibly be left over. This fraction is truncated when the distance value is used in the move algorithm. This truncation error can accumulate when performing several moves over the specified master distance. To eliminate this truncation problem, set the master scale factor (SCLMAS) to 1, or a multiple of 10.

Example: (refer also to the FOLEN examples)

The commands below are a subset of the set-up parameters for an application in which axis 1 is following the encoder input on axis #3 at a 1-to-1 ratio.

```
SCALE1          ; Enable parameter scaling
SCLA25000       ; Set slave acceleration scale factor to 25000 for axis 1
SCLV25000       ; Set slave velocity scale factor to 25000 for axis 1
SCLD25000       ; Set slave distance scale factor to 25000 for axis 1
SCLMAS4000      ; Set master scale factor to 4000 for axis 1
FOLMAS31        ; Axis 1 using encoder input #3 as master
FOLRN1          ; Set slave-to-master Following ratio numerator to 1
                ; (scaled by SCLD)
FOLRD1          ; Set slave-to-master Following ratio denominator to 1.
                ; This sets the ratio to 1:1 (scaled the SCLMAS).
                ; The actual ratio in counts = 25000 to 4000 = 6.25 slave counts
                ; per master count.
```

SCLV — Velocity Scale Factor

Type	Scaling
Syntax	<!><@><a>SCLV<i>,<i>,<i>,<i>
Units	i = steps/unit
Range	1 - 999,999
Default	Steppers: 25000 Servos: Depends on feedback source (4000 if encoder, 4096 if resolver, 819 if ANI, 432 if LDT)
Response	SCLV: *SCLV25000,25000,25000,25000 1SCLV: *1SCLV25000
See Also	EPMV, FMAXV, HOMV, HOMVF, JOGVH, JOGVL, JOYVH, JOYVL, SCALE, SCLA, SCLD, SFB, SSV, TSTAT, V

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Velocity Scale Factor (SCLV) command internally multiplies all velocity (EPMV, HOMV, HOMVF, JOGVH, JOGVL, JOYVH, JOYVL, SSV, V) values by the velocity scale factor value, as long as scaling (SCALE) is enabled. Since the units are steps/unit, all velocities will thus be internally represented in steps/sec.

Steppers: If scaling is enabled (SCALE1), the entered velocity values are internally multiplied by the velocity scaling factor (SCLV) to convert user units/sec to motor steps/sec. The entered values are always in reference to motor steps, not encoder steps, regardless of the ENC command setting.

If scaling is disabled (SCALEØ), all velocity values are entered in motor revs/sec; these values are internally multiplied by the drive resolution (DRES) value to obtain velocity values in motor steps/sec for the motion trajectory calculations.

Servos: If scaling is enabled (SCALE1), the entered velocity values are internally multiplied by the velocity scaling factor (SCLV) to convert user units/sec to encoder, resolver, LDT, or ANI steps/sec.

If scaling is disabled (SCALEØ), all velocity values are entered in encoder or resolver revs/sec, LDT inches/sec, or ANI volts/sec; encoder/resolver and LDT values are internally multiplied by the encoder resolution (ERES) value or LDT resolution (LDTRES) value to obtain velocity values in steps/sec for the motion trajectory calculations.

As the velocity scaling factor (SCLV) changes, the resolution of the velocity commands and the number of positions to the right of the decimal point also change (see table below). A velocity value with greater resolution than allowed will be truncated. For example, if scaling is set to SCLV1Ø, the V1.9999 command would be truncated to V1.9.

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

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

Max. Velocity for Stepper Products		**Max. Velocity for Servo Products** (Servos: determined by feedback source selected for axis #1)	
$\frac{\left(\frac{8{,}000{,}000}{n}\right)}{SCLV}$	*n* = PULSE x 16; If *n* < 5, then *n* is set equal to 5. If *n* > 5, then all fractional parts of *n* are truncated.	Encoder or Resolver Feedback:	$\frac{1000 * ERES}{SCLV}$
		LDT Feedback:	$\frac{1000 * LDTRES}{SCLV}$
		ANI Feedback:	$\frac{1000 * 819}{SCLV}$

NOTE

If scaling is desired within a stored program, you must enable scaling (SCALE1) and define the scaling factors (SCLD, SCLA, and SCLV) prior to defining (DEF), uploading (TPROG), or running (RUN) the program. This allows the 6000 Series product to store, display, and execute the scaled distance, acceleration, and velocity values within the stored program. This can be accomplished by defining all scaling factors via a terminal emulator just before defining or downloading the program; or you can put the scaling factors into a startup (STARTP) program (stand-alone controllers only) or a program that must be run prior to defining or downloading the program.

Example:

```
SCLA25000,25000,1,1 ; Set the acceleration scaling factor for axes 1 and 2 to
                    ; 25000 steps/unit/unit, axes 3 and 4 to 1 step/unit/unit
SCLV25000,25000,1,1 ; Set the velocity scaling factor for axes 1 and 2 to 25000
                    ; steps/unit, axes 3 and 4 to 1 step/unit
SCLD1,1,1,1         ; Set the distance scaling factor for axes 1, 2, 3, and 4 to
                    ; 1 step/unit
SCALE1              ; Enable scaling
DEF prog1           ; Begin definition of program prog1
MA0000              ; Incremental index mode for all axes
MC0000              ; Preset index mode for all axes
A10,12,1,2          ; Set the acceleration to 10, 12, 1, and 2 units/sec/sec
                    ; for axes 1, 2, 3 and 4
V1,1,1,2            ; Set the velocity to 1, 1, 1, and 2 units/sec
                    ; for axes 1, 2, 3 and 4
D100000,1000,10,100 ; Set the distance to 100000, 1000, 10, and 100 units
                    ; for axes 1, 2, 3 and 4
GO1100              ; Initiate motion on axes 1 and 2, 3 and 4 do not move
END                 ; End definition of program prog1
```

SD — Streaming Data

Type	Motion
Syntax	<!><@>SD<i>,<i>,<i>,<i>
Units	(varies—see below)
Range	(varies—see below)
Default	n/a
Response	None
See Also	PULSE, STD, STREAM

Product	Rev
AT6n00	1.1
AT6n50	n/a
610n	n/a
615n	n/a
620n	n/a
625n	n/a
6270	n/a

While in the Streaming Mode (STREAM1 or STREAM2), you can use Streaming Data (SD) commands to change distance or velocity values and enable certain streaming functions (see table below). Each data or function assignment (<i>) represents one *datapoint*. As many as four datapoints are possible per SD command—one for each axis.

Function of the SD Command	Range for <i> (*Datapoint*)
Distance or velocity data	0 to ±32767
Wait for input pattern *	1bbbbbbbb (b = 0 or 1)
Set outputs *	2bbbbbbbb (b = 0 or 1)
Set mask *	3bbbbbbbb (b = 0 or 1)
Set loop *	400000000 to 499999999
End loop *	500000000
Terminate loop *	600000000
Exit streaming mode *	700000000
Set negative-direction	800000000

* These functions must be assigned in the SD data field that corresponds to the first streaming axis. For example, if you enabled the Distance Streaming Mode for axes 3 & 4 (STREAM,,1,1), the Set Loop datapoint 400000012 must be entered in the third axis' data field (SD,,400000012).

CAUTION

In both streaming modes, the SD commands are executed in the motion trajectory update. Because of processing time constraints, error checking is minimal. For instance, a 2 in a field designated for a 1 or 0 may turn on unexpected outputs. Entering data greater than the maximum distance or frequency will cause unexpected motor positioning. If incorrect data or no data is detected, the data is ignored and the last velocity or distance value is executed.

Distance or Velocity Data (0 to ±32767):

When in the Distance Streaming Mode (STREAM1), the SD command value (<i>) is the number of motor steps traveled per time interval set by the Streaming Interval (STD) command. The direction is determined by the sign of the data; that is, + for positive-direction and - for negative-direction. The maximum distance per update is determined by the following equation:

Maximum distance per update interval = *Streaming interval* ∗ *Maximum pulse rate* - 1

Where: The *Maximum distance per update interval* is expressed in motor steps. The *Streaming interval* is determined by the STD command setting and expressed in seconds. The *Maximum pulse rate* is determined by the PULSE command setting and expressed in steps/second (**be sure to set the PULSE value the same on all axes involved in streaming**).

For example, an STD command value of 20 ms (= 0.020 seconds) and a PULSE command value of 1 µs (max. pulse rate = 500,000 steps/second) provides the following equation:

Maximum distance per update interval = 0.020 ∗ 500000 - 1 = 9999 motor steps

When in the Velocity Streaming Mode (STREAM2), the SD command value (<i>) is related to the frequency output per streaming interval set by the STD command:

$$\text{SD } value = \text{Step output frequency} * \frac{32767}{\text{Maximum pulse rate}}$$

Where: The *Step output frequency* is expressed in pulses/second. The *Maximum pulse rate* is determined by the PULSE command setting and expressed in steps/second (**be sure to set the PULSE value the same on all axes involved in streaming**).

For example, a desired frequency of 1000 and a PULSE command value of 1 µs (max. pulse rate = 500,000 steps/second) provides the following equation:

$$\text{SD } value = 1000 * \frac{500000}{32767} = \text{SD15259}$$

NOTE

When in the distance or velocity streaming mode, the last SD data point output will continue to be output on each succeeding update, unless a new SD data point is received. If you have all of your SD data points in a program that is contained in the AT6n00, this will pose no problem; however, if you are sending each individual SD data point from an external program *on the fly*, be sure to not exceed the update period you specified with the STD command.

NOTE

The command examples provided below show SD commands used in the Distance Streaming Mode (STREAM1). This is done simply to demonstrate the use of the commands. These command examples could be used just as easily in the Velocity Streaming Mode (STREAM2).

Wait for Input Pattern (SD1bbbbbbbb):

Where b = 1 for ON and 0 for OFF. Input 1 is the leftmost b, input 8 the rightmost b (inputs 1 through 8 are always used, regardless of their assigned function). This SD data command (SD1bbbbbbbb), in combination with an input mask (SD3bbbbbbbb), determine the input pattern for which to wait. If the mask is omitted, the pattern will be determined solely by the wait for input SD data command (SD1bbbbbbbb). While waiting for the input state to be true, the last velocity is continually output. If zero velocity is desired during the wait, set the SD data point prior to the SD1bbbbbbbb to 0 or 800000000. All streaming axes will wait for the input function. **Place the wait for input function in the first streaming axis field.**

Only one SD1bbbbbbbb command is allowed per steaming interval, others are ignored (i.e., two SD1bbbbbbbb commands cannot be back to back).

Example:

```
STREAM1        ; Enable Streaming Mode on axis 1
SD55           ; Move 55 steps positive-direction
SD0            ; Move 0 steps positive-direction
SD111000000    ; Wait for inputs 1 & 2 to become active, while inputs 3, 6,
               ; 7, & 8 are inactive. Inputs 4 & 5 can be either state due
               ; to the subsequent mask. Note that the Mask (SD3bbbbbbbb)
               ; command must always follow the Wait for Input Pattern
               ; (SD1bbbbbbbb) command.
SD300011000    ; Mask inputs 4 and 5. Input Pattern = 110XX000.
SD150          ; Move 150 steps positive-direction
SD800000000    ; Move 0 steps negative-direction
SD111011000    ; Wait for inputs 1, 2, 4, & 5 to become active, while
               ; inputs 3, 6, 7, & 8 are inactive.
SD-150         ; Move 150 steps negative-direction
SD700000000    ; Exit Streaming Mode
```

Set Output State (SD2bbbbbbbb**):**

Where b = 1 for ON and 0 for OFF. Output 1 is the leftmost b, output 8 the rightmost b (assuming outputs 1 through 8 are defined as *programmable* outputs—see OUTFNC-A). This SD data command (SD2bbbbbbbb), in combination with an output mask (SD3bbbbbbbb), determines the output pattern. If the mask is omitted, the pattern will be determined solely by the Set Output State SD data command (SD2bbbbbbbb). **Place the Set Output State in the first streaming axis field.**

Only one SD2bbbbbbbb command is allowed per streaming interval, others are ignored (i.e., two SD2bbbbbbbb commands cannot be back to back).

Example:

```
STREAM1        ; Enable Streaming Mode on axis 1
SD55           ; Move 55 steps positive-direction
SD211000001    ; Turn on outputs 1,2, & 8. Turn off outputs 3 & 7.
               ; Outputs 4, 5, & 6 will remain unchanged due to the mask.
               ; The mask command must always follow the output state command.
SD300011100    ; Mask outputs 4,5, and 6. Output Pattern = 110XXX01.
SD150          ; Move 150 steps positive-direction
SD200111110    ; Turn off outputs 1,2, & 8. Turn on outputs 3, 4, 5, 6, & 7.
SD800000000    ; Move 0 steps negative-direction (sets negative-direction)
SD-150         ; Move 150 steps negative-direction
SD700000000    ; Exit Streaming Mode
```

Mask for Inputs and Outputs (SD3bbbbbbbb**):**

Where b = 1 for a masked bit and 0 for an enabled bit. Bit 1 is the leftmost b, bit 8 the rightmost b. The mask, in conjunction with the input and output SD data values (SD1bbbbbbbb and SD2bbbbbbbb), is used to determine input and output patterns. If omitted, the mask defaults to SD300000000, which enables all eight bits. **Place the mask in the first streaming axis field.**

NOTE: The Mask (SD3bbbbbbbb) must follow the input (SD1bbbbbbbb) or output (SD2bbbbbbbb) commands.

Example:

```
STREAM,,1,1    ; Enable Streaming Mode on axes 3 & 4
SD,,211000001  ; Turn on outputs 1,2, & 8. Turn off outputs 3 & 7.
               ; Outputs 4, 5, & 6 will remain unchanged due to the mask.
SD,,300011100  ; Mask outputs 4,5, and 6. Output Pattern = 110XXX01.
SD,,700000000  ; Exit Streaming Mode
```

Loop (SD400000000 + i):

Where i sets the loop count value. All the commands between the loop data point and the end loop data point are repeated the number of times indicated by i. A value of 0 indicates an infinite loop. A maximum of 60 SD values per axis are allowed inside the loop. The loop can be terminated by the !SD600000000 stop loop data command. Loops cannot be nested. All streaming axes will be in the loop. **Place the loop command in the first streaming axis field.**

Only one loop can be executed per streaming interval (i.e., two SD4nnnnnnnn commands cannot be back to back).

Example:

```
STREAM,1,1     ; Enable Streaming Mode on axes 2 & 3
SD,400000012   ; Loop 12 times
SD,50,50       ; Move 50 steps per streaming interval on axes 2 and 3
SD,500000000   ; End loop
SD,700000000   ; Exit Streaming Mode
```

End Loop (SD500000000):

Used in conjunction with the loop SD value, the End Loop (SD500000000) command establishes the loop demarcations. **Place the end loop command in the first streaming axis field.**

Example:

```
STREAM1        ; Enable Streaming Mode on axis 1
SD400002125    ; Loop 2125 times
SD58           ; Move 58 steps per streaming interval on axis 1
SD500000000    ; End loop
SD700000000    ; Exit Streaming Mode
```

Terminate Loop (SD600000000):

If in a streaming loop, the Terminate Loop (!SD600000000) command stops the loop at the end of its next iteration.

Example:

```
STREAM1       ; Enable Streaming Mode on axis 1
SD400000000   ; Loop forever (infinite)
SD58          ; Move 58 steps per streaming interval
SD500000000   ; End loop
SD700000000   ; Exit Streaming Mode
```

The SD58 command will be executed infinitely. !SD600000000 will terminate the loop.

Terminate Streaming (SD700000000):

The Terminate Streaming (SD700000000) command exits the streaming mode for all axes. The STREAM command value is set to a 0. **This command must be included in all streaming programs.**

Negative- Direction Change (SD800000000):

This command sets the direction bit for motion in the negative-direction at zero velocity. *Some motor drives require a set-up time for the direction prior to receiving pulses.* To set the direction bit for motion in the positive-direction, issue the SD0 command.

Example:

```
STREAM1       ; Enable Streaming Mode on axis 1
SD0           ; Move 0 steps, also set positive-direction
SD20          ; Move 20 steps positive-direction
SD55          ; Move 55 steps positive-direction
SD800000000   ; Move 0 steps, also set negative-direction
SD-52         ; Move 52 steps negative-direction
SD700000000   ; Exit Streaming Mode
```

Sample Program Demonstrates Streaming Data:

Example:

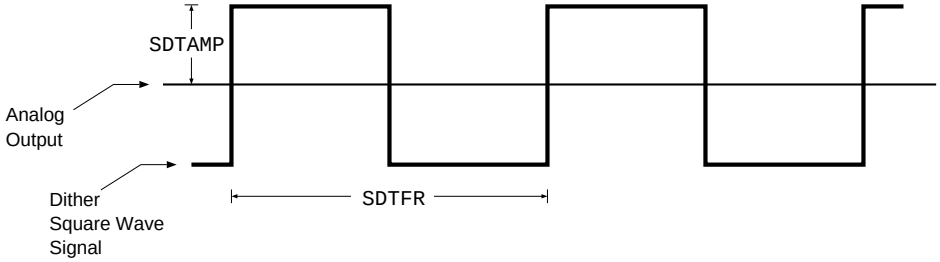
```
DEF SAMPLE    ; Begin definition of program sample
@PULSE1       ; Set pulse width to 1 microsecond
STD20         ; Set streaming interval to 20 milliseconds
@STREAM1      ; Set distance streaming mode on all axes
@SD75         ; Travel 75 steps in 20 milliseconds on all axes
SD101100000   ; Wait for input pattern
SD300001111   ; Set mask for input pattern; input pattern = 0110XXXX
SD200001111   ; Set outputs
SD311110000   ; Set mask for outputs; output pattern = XXXX1111
SD400000200   ; Loop 200 times
@SD12         ; Travel 12 steps (+ direction) in 20 milliseconds on all axes
@SD32         ; Travel 32 steps (+ direction) in 20 milliseconds on all axes
@SD58         ; Travel 58 steps (+ direction) in 20 milliseconds on all axes
@SD88         ; Travel 88 steps (+ direction) in 20 milliseconds on all axes
SD500000000   ; End loop
SD201010001   ; Set outputs
SD300001110   ; Set mask for outputs; output pattern = 0101XXX1
@SD700000000  ; Exit streaming mode
WAIT(MOV=b0000)  ; Wait for motion to stop on all axes
END           ; End definition of program sample
SAMPLE        ; Execute program sample
```

SDTAMP Dither Amplitude

		Product	Rev
Type	Servo	AT6n00	n/a
Syntax	<!><@><a>SDTAMP<r>,<r>,<r>,<r>	AT6n50	1.0
Units	r = peak volts	610n	n/a
Range	0.000 - 2.000	615n	1.0
Default	0.000	620n	n/a
Response	SDTAMP *SDTAMP0.000,0.000,0.000,0.000 1SDTAMP *1SDTAMP0.000	625n	3.0
See Also	[DAC], SDTFR, SOFFS, SSFR, TDAC	6270	1.0

Use the SDTAMP command to select the amplitude of a square wave dither signal superimposed on the analog output (DAC).

Dither is a square-wave signal added to the servo controller's analog output signal and can be used to keep a hydraulic valve moving slightly for the purpose of reducing stiction (see illustration below). The SDTAMP and SDTFR commands are used to select the amplitude and frequency, respectively.



The SDTAMP command selects the amplitude of the dither signal in peak volts (see illustration). The SDTFR command selects the frequency ratio of the dither.

The actual dither frequency is determined by the ratio of the servo sampling frequency (affected by the SSFR and INDAX command settings) and the SDTFR value. For example, if the SSFR value is 4 and the INDAX value is 2, the servo sampling rate is 2500 samples per second. Then, at SSFR4, an SDTFR value of 46 (default setting) would yield a 54.3 Hz dither frequency (2500/46 = 54.3). With an SDTFR command setting of 46, a positive voltage (SDTAMP) is added during 23 servo updates and a negative voltage is added during the next 23 servo updates.

Example:

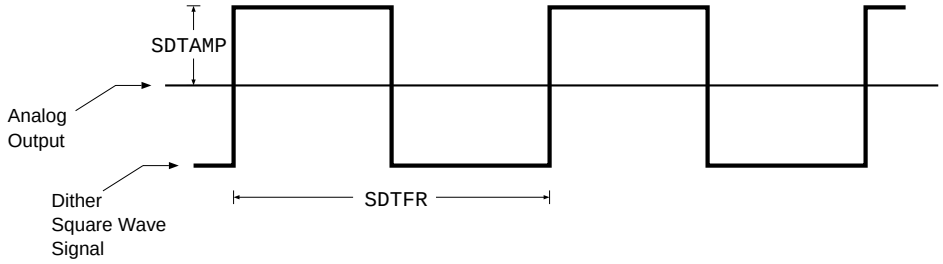
```
INDAX2          ; Use two axes of motion
SDTAMP.1,.1     ; Dither amplitude for both axes is 0.1V peak
SSFR8           ; Select sampling frequency. The table in the SSFR command
                ; description shows that the sampling rate is 2700 servo
                ; samples/second.
SDTFR100,50     ; Dither frequency for axis 1 is 27 Hz (2700/100 = 27),
                ; and Dither frequency for axis 2 is 54 Hz (2700/50 = 54)
```

SDTFR Dither Frequency Ratio

		Product	Rev
Type	Servo	AT6n00	n/a
Syntax	<!><@><a>SDTFR<i>,<i>,<i>,<i>	AT6n50	1.0
Units	i = Servo samples per period	610n	n/a
Range	2 - 1000 (use even values only—odd values will be rounded down)	615n	1.0
Default	46	620n	n/a
Response	SDTFR *SDTFR46,46,46,46 1SDTFR *1SDTFR46	625n	3.0
See Also	[DAC], SDTAMP, SOFFS, SSFR, TDAC	6270	1.0

Use the SDTFR command to select the frequency ratio of a square wave dither signal superimposed on the analog output (DAC).

Dither is a square-wave signal added to the servo controller's analog output signal can be used to keep a hydraulic valve moving slightly for the purpose of reducing stiction (see illustration below). The SDTAMP and SDTFR commands are used to select the amplitude and frequency, respectively.



The SDTAMP command selects the amplitude of the dither signal in peak volts (see illustration). The SDTFR command selects the frequency ratio of the dither.

The actual dither frequency is determined by the ratio of the servo sampling frequency (affected by the SSFR and INDAX command settings) and the SDTFR value. For example, if the SSFR value is 4 and the INDAX value is 2, the servo sampling rate is 2500 samples per second. Then, at SSFR4, an SDTFR value of 46 (default setting) would yield a 54.3 Hz dither frequency (2500/46 = 54.3). With an SDTFR command setting of 46, a positive voltage (SDTAMP) is added during 23 servo updates and a negative voltage is added during the next 23 servo updates.

Example:

```
INDAX2          ; Use two axes of motion
SDTAMP.1,.1     ; Dither amplitude for both axes is 0.1V peak
SSFR8           ; Select sampling frequency. The table in the SSFR command
                ; description shows that the sampling rate is 2700 servo
                ; samples/second.
SDTFR100,50     ; Dither frequency for axis 1 is 27 Hz (2700/100 = 27),
                ; and Dither frequency for axis 2 is 54 Hz (2700/50 = 54)
```

[SEG] Number of Free Segment Buffers

Type	Compiled Motion; Assignment or Comparison
Syntax	<!>[SEG]
Units	n/a
Range	n/a
Default	n/a
Response	n/a
See Also	MEMORY, [SS], TDIR, TMEM, TSEG, TSS

Product	Rev
AT6n00	4.1
AT6n50	4.1
610n	4.0
615n	4.1
620n	4.1
625n	4.1
6270	4.1

The Number of Free Segment Buffers [SEG] command is used to assign the number of free memory segment buffers in compiled memory to a variable (VAR), or to make a comparison against another value.

System status bit (see TSSF, TSS, and SS) 29 to set when the compiled memory is 75% full, and bit 30 is set if the compiled memory is 100% full.

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

SFB — Select Servo Feedback Source

Type	Controller Configuration or Servo
Syntax	<@><a>SFB<i>,<i>,<i>,<i>
Units	i = feedback source identifier
Range	i = 0 (open loop, disable all gains), 1 (encoder), 2 (ANI input), 3 (LDT), or 4 (resolver)
Default	1 (3 for 6270 only, 4 for 615n only)
Response	SFB *SFB1,1,1,1 1SFB *1SFB3
See Also	[ANI], [CA], ERES, [FB], [LDT], LDTGRD, LDTRES, OUTPA, OUTPB, [PANI], [PCA], [PCE], [PCL], [PE], PSET, SCALE, SCLD, SGAF, SGAFN, SGI, SGILIM, SGIN, SGP, SGPN, SGV, SGVF, SGVFN, SGVN, SMPER, SOFFS, SOFFSN, TANI, TCA, TFB, TLDT, TPANI, TPE

Product	Rev
AT6n00	n/a
AT6n50	1.0
610n	n/a
615n	1.0
620n	n/a
625n	3.0
6270	1.0

Use the SFB command to select the servo feedback source to be used by each axis.

Product	Options	Associated Connectors	Measurement*	Resolution Command
615n	1—External Encoder	External Encoder	Encoder counts	ERES command
	2—ANI input (option)	Auxiliary	ADC counts	n/a
	4—Internal Resolver	Internal	Resolver counts	ERES command
AT6n50 & 625n	1—Encoder	**ENCODER 1** – **ENCODER 4**	Encoder counts	ERES command
	2—ANI input (ANI option only)	AT6n50 & OEM625n: External ANI board	ADC counts	n/a
		625n: Pin 7 on **DRIVE 1** & **DRIVE 2**		
6270	1—Encoder (axis 1 only)	**ENCODER 1** connector only	Encoder counts	ERES command
	2—ANI input (6270-ANI only)	Pin 7 on **DRIVE 1** & **DRIVE 2**	ADC counts	n/a
	3—LDT	**LDT 1** & **LDT 2**	LDT counts	LDTRES command **

***** With scaling enabled (SCALE1), LDT, encoder and ANI feedback is scaled by the SCLD value,
****** Use the LDTGRD (gradient) command to compensate for gradient variations between LDTs.

NOTE

Parameters for scaling (SCLA, SCLD, etc.) , tuning gains (SGI, SGP, etc.), position offset (PSET) and maximum position error (SMPER) are specific to the feedback source currently selected with the last SFB command.

If your application requires switching between feedback sources for the same axis, then for each feedback source, you must issue the SFB command and then enter the scaling, gains, PSET and SMPER commands specific to that feedback source.

The feedback source can be changed only if motion is not in progress. When the feedback source is changed, the new setpoint will be determined by taking the new feedback source's value and adding any existing position error. Changing the source will disable the Output On Position commands (OUTPn).

USING SFBØ

Setting the SFB command value to zero has these effects:

- **WARNING**: The end-of-travel limits are disabled. Make sure that it is safe to operate without end-of-travel limits before using SFBØ.
- Gain values (SGILIM, SGAF, SGAFN, SGI, SGIN, SGP, etc.) set to zero (open-loop operation).
- SMPER value set to zero (position error is allowed to increase without causing a fault.
- Subsequent attempts to change gain values or SMPER will cause an error message ("NOT ALLOWED IF SFBØ")
- SOFFS and SOFFSN set to zero, but allows subsequent servo offset changes to affect motion.
- SSWG set to zero (disables the Setpoint Window Gains feature).
- Disables output-on-position (OUTPA - OUTPD) functions.
- Any subsequent changes to PSET, PSETCLR, SCLD, SCLA, SCLV, SOFFS, and SOFFN are lost when another feedback source is selected.

Recommendation: Use the Disable Drive On Kill more, enabled with the KDRIVE command, so that the controller will shut down the drive if a kill command (e.g., !K) is executed or if a kill input is activated. Keep in mind that shutting down the drive allows the load to freewheel if there is not brake installed.

If you are using a 4 - 20mA signal for feedback, you can convert current to voltage by connecting a resistor between the **ANI** input terminal and the **AGND** terminal. For example, a 500Ω resistor would provide voltage values of 2-10V.

Example:

```
DEF Setup       ; Begin definition of program called setup
DRIVE0          ; Disable (shutdown) axis #1
SFB1            ; Select encoder feedback for axis #1 (subsequent scaling,
                ; gains, and PSET are specific to encoder feedback operation)
ERES4000        ; Set encoder resolution
SCLA4000        ; Set scaling for programming acceleration in revs/sec/sec
SCLV4000        ; Set scaling for programming velocity in revs/sec
SCLD4000        ; Set scaling for programming distance in revs
SGP5            ; Set proportional feedback gain to 5
SGI1            ; Set integral feedback gain to 1
SGV1            ; Set velocity feedback gain to 1
PSET0           ; Set current position as absolute position zero
SFB2            ; Select ANI feedback for axis #1 (subsequent scaling, gains,
                ; and PSET are specific to ANI feedback operation)
SCLA819         ; Set scaling for programming acceleration in volts/sec/sec
SCLV819         ; Set scaling for programming velocity in volts/sec
SCLD819         ; Set scaling for programming distance in volts
SGP1            ; Set proportional feedback gain to 1
SGI0            ; Set integral feedback gain to zero
SGV.5           ; Set velocity feedback gain to 0.5
PSET0           ; Set current position as absolute position zero
SFB1            ; Select encoder feedback for axis #1
END             ; End definition of program called setup
```

SGAF — Acceleration Feedforward Gain

		Product	Rev
Type	SERVO	AT6n00	n/a
Syntax	<!><@><a>SGAF<r>,<r>,<r>,<r>	AT6n50	1.0
Units	r = microvolts/step/sec/sec	610n	n/a
Range	0.00000000 - 2800000.00000000	615n	1.0
Default	0	620n	n/a
Response	SGAF: *SGAF0,0,0,0 1SGAF: *1SGAF0	625n	1.0
See Also	SFB, SGAFN, SGENB, SGI, SGP, SGSET, SGV, SGVF, SSWG, TGAIN, TSGSET	6270	1.0

Use the Acceleration Feedforward Gain (SGAF) command to set the gain for the acceleration feedforward term in the servo control algorithm. Introducing acceleration feedforward control improves *position tracking performance* when the system is commanded to accelerate or decelerate.

The SGAF value is multiplied by the *commanded acceleration* (calculated by the 6000 controller's DSP move profile routine) to produce the control signal.

Acceleration feedforward control can improve the performance of contouring and linear interpolation applications, as well as reduce the time required to reach the commanded velocity. *However, if your application only requires point-to-point moves, acceleration feedforward control is not necessary (leave the SGAF command setting at zero—default).*

Acceleration feedforward control does not affect the servo system's stability, nor does it have any effect at constant velocity or at steady state.

NOTE

The SGAF command is specific to the current feedback source (selected with the last SFB command). Therefore, if your application requires switching between feedback sources for the same axis, then for each feedback source, you must select the feedback source with the appropriate SFB command and then issue the SGAF command with the gain values specific to the selected feedback source.

For more information on servo tuning and how the acceleration feedforward gain affects performance, refer to your product's *Installation Guide* or to the *Servo Tuner User Guide*.

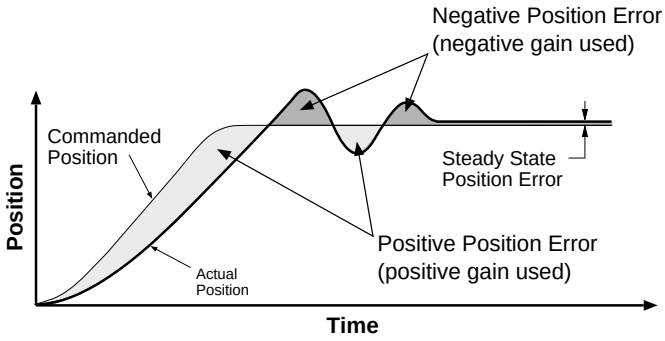
Example:

```
SGAF0.5555,43.554,0,0      ; Set the acceleration feedforward for axes 1 and 2
```

SGAFN — Acceleration Feedforward Gain — Negative

		Product	Rev
Type	Servo	AT6n00	n/a
Syntax	<!><@><a>SGAFN<r>,<r>	AT6n50	n/a
Units	r = microvolts/step/sec/sec	610n	n/a
Range	0.00000000 to 2800000.00000000	615n	n/a
Default	0	620n	n/a
Response	SGAFN *SGAFN0,0 1SGAFN *1SGAFN0	625n	n/a
See Also	SFB, SGAF, SGENB, SGI, SGIN, SGP, SGPN, SGSET, SGV, SGVF, SGVFN, SGVN, SOFFS, SOFFSN, SSWG, TGAIN, TSGSET	6270	3.0

Use the SGAFN command to set the gain of the acceleration feedforward term in the control algorithm when the position error is negative (see illustration below). The controller automatically switches between the positive and negative gains to correlate with the positive or negative position error. This is particularly useful when controlling hydraulic cylinders in which the different surface areas on each side of the piston react differently with the same gain settings.



Negative Gain used = SGAFN

Positive Gain used = SGAF

Introducing acceleration feedforward control improves *position tracking performance* when the system is commanded to accelerate or decelerate.

The SGAFN value changes with (tracks) the corresponding positive gain value (SGAF) until a SGAFN command is executed. After the SGAFN command is executed, separate positive and negative gain commands must be used. To re-establish the default mode where SGAFN tracks SGAF, issue the SGAFN command with a minus sign (-) in the command field for the affected axis (e.g., SGAFN,- restores axis 2 to the default mode).

The SGAFN value is multiplied by the *commanded acceleration* (calculated by the 6000 controller's DSP move profile routine) to produce the control signal. Acceleration feedforward control does not affect the servo system's stability, nor does it have any effect at constant velocity or at steady state.

Acceleration feedforward control can improve the performance of contouring and linear interpolation applications, as well as reduce the time required to reach the commanded velocity. *However, if your application only requires point-to-point moves, acceleration feedforward control is not necessary (leave the SGAFN command at its default setting).*

NOTE

The SGAFN command is specific to the current feedback source (selected with the last SFB command). Therefore, if your application requires switching between feedback sources for the same axis, then for each feedback source, you must select the feedback source with the appropriate SFB command and then issue the SGAFN command with the gain values specific to the selected feedback source.

For more information on servo tuning and how the acceleration feedforward gain affects performance, refer to your product's *Installation Guide* or to the *Servo Tuner User Guide*.

SGENB — Enable a Servo Gain Set

Type	SERVO
Syntax	<!><@><a>SGENB<i>,<i>,<i>,<i>
Units	i = gain set identification number (see SGSET command)
Range	1 - 5
Default	n/a
Response	n/a
See Also	SFB, SGAF, SGAFN, SGI, SGILIM, SGIN, SGP, SGPN, SGSET, SGV, SGVF, SGVFN, SGVN, SOFFS, SOFFSN, SSWG, TGAIN, TSGSET

Product	Rev
AT6n00	n/a
AT6n50	1.0
610n	n/a
615n	1.0
620n	n/a
625n	1.0
6270	1.0

This command allows you to enable any combination of the five gain sets to any combination of axes. The gain sets are set with the SGSET command. A gain set can be enabled during motion at any specified point in the profile, or when not in motion. For example, you could use one set of gain parameters for the constant velocity portion of the profile, and when you approach the target position a different set of gains can be enabled. 6270: In a hydraulic application a different set of gains can be enabled depending on the direction of motion.

NOTE

The tuning gains in a given gain set are specific to the feedback source that was in use (selected with the last SFB command) at the time the gains were established with the respective gain commands (SGI, SGP, etc.). Make sure that the gain set you enable is appropriate to the feedback source you are using at the time.

For more information on servo tuning, refer to your product's *Installation Guide* or to the *Servo Tuner User Guide*.

Example:

```
SGP5,5,10,10    ; Sets the gains for the proportional gain
SGI.1,.1,0,0    ; Sets the gains for the integral gain
SGV50,60,0,0    ; Sets the gains for the velocity gain
SGVF5,6,10,11   ; Sets the gains for the velocity feedforward gain
SGAF0,0,0,0     ; Sets the gains for the acceleration feedforward gain
SGSET3          ; Assign SGP, SGI, SGV, SGVF, & SGAF gains to servo gain set 3
SGP75,75,40,40  ; Sets the gains for the proportional gain
SGI5,5,5,7      ; Sets the gains for the integral gain
SGV1,.45,2,2    ; Sets the gains for the velocity gain
SGVF0,8,0,9     ; Sets the gains for the velocity feedforward gain
SGAF18,20,22,24 ; Sets the gains for the acceleration feedforward gain
SGSET1          ; Assign SGP, SGI, SGV, SGAF, & SGVF gains to servo gain set 1
SGENB1,3,3,1    ; Enables gain set 1 gains on axis 1 &4; enables gain set 3 on
                ; axis 2 & 3
TGAIN           ; Displays the current value for all gains.  Example response:
                ;   *SGP75,5,10,40
                ;   *SGI5,.1,0,7
                ;   *SGV1,60,0,2
                ;   *SGVF0,6,10,9
                ;   *SGAF18,0,0,24
```

SGI — Integral Feedback Gain

Type	SERVO
Syntax	<!><@><a>SGI<r>,<r>,<r>,<r>
Units	r = millivolts/step * sec
Range	0.00000000 - 2800000.00000000
Default	0
Response	SGI: *SGI0,0,0,0 1SGI: *1SGI0
See Also	SFB, SGAF, SGENB, SGILIM, SGIN, SGP, SGSET, SGV, SGVF, SSWG, TGAIN, TSGSET

Product	Rev
AT6n00	n/a
AT6n50	1.0
610n	n/a
615n	1.0
620n	n/a
625n	1.0
6270	1.0

Use the Integral Gain (SGI) command to set the gain of the integral term in the control algorithm. The primary function of the integral gain is to reduce or eliminate final position error (e.g., due to friction, gravity, etc.) and improve system accuracy during motion. If a position error exists (commanded position not equal to actual position—see TPER command), this control signal will ramp up until it is high enough to overcome the friction and drive the motor toward its commanded position. *If acceptable position accuracy is achieved with proportional gain (SGP), then the integral gain (SGI) need not be used.*

If the integral gain is set too high relative to the other gains, the system may become oscillatory or unstable. The integral gain can also cause excessive position overshoot and oscillation if an appreciable position error has persisted long enough during the transient period (time taken to reach the position setpoint); this effect can be reduced by using the SGILIM command to limit the integral term windup.

NOTE

The SGI command is specific to the current feedback source (selected with the last SFB command). Therefore, if your application requires switching between feedback sources for the same axis, then for each feedback source, you must select the feedback source with the appropriate SFB command and then issue the SGI command with the gain values specific to the selected feedback source.

For more information on servo tuning, refer to your product's *Installation Guide* or to the *Servo Tuner User Guide*.

Example:
```
SGI15,14.5      ; Set the integral gain for axes 1 and 2
```

SGILIM Integral Windup Limit

Type	SERVO
Syntax	<!><@><a>SGILIM<r>,<r>,<r>,<r>
Units	r = volts
Range	0 - 65535
Default	200
Response	SGILIM: *SGILIM200,200,200,200 1SGILIM: *1SGILIM200
See Also	SFB, SGENB, SGI, SSWG, TGAIN, TSGSET

Product	Rev
AT6n00	n/a
AT6n50	1.0
610n	n/a
615n	1.0
620n	n/a
625n	1.0
6270	1.0

If integral control (SGI) is used and an appreciable position error has persisted long enough during the transient period (time taken to reach the setpoint), the control signal generated by the integral action can end up too high and saturate to the maximum level of the controller's analog control signal output. This phenomenon is called *integrator windup*.

After windup occurs, it will take a while before the integrator output returns to a level within the limit of the controller's output. Such a delay causes excessive position overshoot and oscillation. Therefore, the integral windup limit (SGILIM) command is provided for you to set the absolute limit of the integral and, in essence, turn off the integral action as soon as it reaches the limit; thus, position overshoot and oscillation can be reduced.

NOTE

The SGILIM command is specific to the current feedback source (selected with the last SFB command). Therefore, if your application requires switching between feedback sources for the same axis, then for each feedback source, you must select the feedback source with the appropriate SFB command and then issue the SGILIM command with the gain values specific to the selected feedback source.

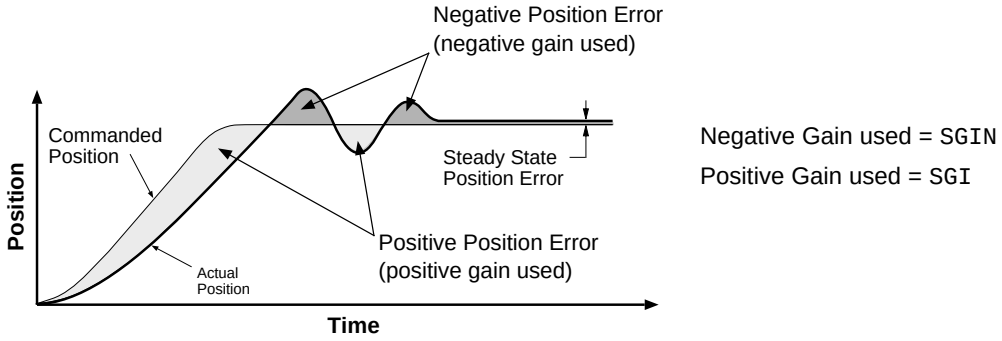
For more information on servo tuning, refer to your product's *Installation Guide* or to the *Servo Tuner User Guide*.

Example:
```
SGI44,43,55,0      ; Sets the integral gain term
SGILIM15,15,15,15  ; Sets the integral windup limit on the integral gain term
```

SGIN — Integral Feedback Gain — Negative

		Product	Rev
Type	Servo	AT6n00	n/a
Syntax	<!><@><a>SGIN<r>,<r>	AT6n50	n/a
Units	r = millivolts/step * sec	610n	n/a
Range	0.00000000 to 2800000.00000000	615n	n/a
Default	0	620n	n/a
Response	SGIN *SGIN0,0 1SGIN *1SGIN0	625n	n/a
See Also	SFB, SGAF, SGAFN, SGENB, SGI, SGILIM, SGP, SGPN, SGSET, SGV, SGVF, SGVFN, SGVN, SOFFSN, SSWG, TGAIN, TSGSET	6270	3.0

Use the SGIN command to set the gain of the integral term in the control algorithm when the position error is negative (see illustration below). The controller automatically switches between the positive and negative gains to correlate with the positive or negative position error. This is particularly useful when controlling hydraulic cylinders in which the different surface areas on each side of the piston react differently with the same gain settings.



The primary function of the integral gain is to reduce or eliminate final position error (e.g., due to friction, gravity, etc.) and improve system accuracy during motion. If a position error exists (commanded position not equal to actual position—see TPER command), this control signal will ramp up until it is high enough to overcome the friction and drive the motor toward its commanded position. *If acceptable position accuracy is achieved with proportional gain (SGP), then the integral gain (SGI) need not be used.*

The SGIN value changes with (tracks) the corresponding positive gain value (SGI) until a SGIN command is executed. After the SGIN command is executed, separate positive and negative gain commands must be used. To re-establish the default mode where SGIN tracks SGI, issue the SGIN command with a minus sign (-) in the command field for the affected axis (e.g., SGIN,- restores axis 2 to the default mode).

If the integral gain is set too high relative to the other gains, the system may become oscillatory or unstable. The integral gain can also cause excessive position overshoot and oscillation if an appreciable position error has persisted long enough during the transient period (time taken to reach the position setpoint); this effect can be reduced by using the SGILIM command to limit the integral term windup.

NOTE

The SGIN command is specific to the current feedback source (selected with the last SFB command). Therefore, if your application requires switching between feedback sources for the same axis, then for each feedback source, you must select the feedback source with the appropriate SFB command and then issue the SGIN command with the gain values specific to the selected feedback source.

For more information on servo tuning, refer to your product's *Installation Guide* or to the *Servo Tuner User Guide*.

SGP — Proportional Feedback Gain

		Product	Rev
Type	SERVO		
Syntax	<!><@><a>SGP<r>,<r>,<r>,<r>	AT6n00	n/a
Units	r = millivolts/step	AT6n50	1.0
Range	0.00000000 - 2800000.00000000	610n	n/a
Default	0.5	615n	1.0
Response	SGP: *SGP0.5,0.5,0.5,0.5 1SGP: *1SGP0.5	620n	n/a
		625n	1.0
See Also	SFB, SGAF, SGENB, SGI, SGPN, SGSET, SGV, SGVF, SSWG, TGAIN, TSGSET	6270	1.0

This command allows you to set the gain of the proportional term in the servo control algorithm. The output of the proportional term is proportional to the difference between the commanded position and the actual position read from the feedback device. The primary function of the proportional term is to stabilize the system and speed up the response. It can also be used to reduce the steady state position error.

When the proportional gain (SGP) is used alone (i.e., the other gain terms are set to zero), setting this gain too high can cause the system to become oscillatory, underdamped, or even unstable.

NOTE

The SGP command is specific to the current feedback source (selected with the last SFB command). Therefore, if your application requires switching between feedback sources for the same axis, then for each feedback source, you must select the feedback source with the appropriate SFB command and then issue the SGP command with the gain values specific to the selected feedback source.

For more information on servo tuning and how the proportional gain affects tuning, refer to your product's *Installation Guide* or to the *Servo Tuner User Guide*.

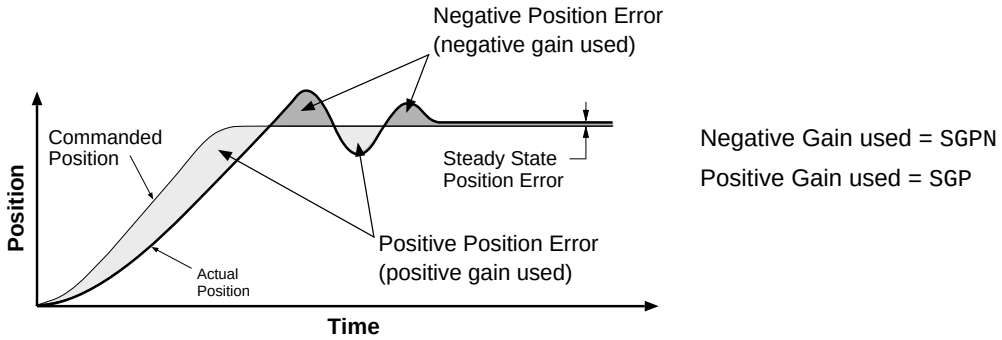
Example:

```
SGP10,4.22233,2.22,.0445245      ; Sets the proportional gain of all axes
```

SGPN — Proportional Feedback Gain — Negative

		Product	Rev
Type	Servo		
Syntax	<!><@><a>SGPN<r>,<r>	AT6n00	n/a
Units	r = millivolts/step	AT6n50	n/a
Range	0.00000000 to 2800000.00000000	610n	n/a
Default	0.5	615n	n/a
Response	SGPN *SGPN0.5,0.5 1SGPN *1SGPN0.5	620n	n/a
		625n	n/a
See Also	SFB, SGAF, SGAFN, SGENB, SGI, SGIN, SGP, SGSET, SGV, SGVF, SGVFN, SGVN, SOFFSN, SSWG, TGAIN, TSGSET	6270	3.0

The SGPN command allows you to set the gain of the proportional term in the servo control algorithm when the position error is negative (see illustration below). The controller automatically switches between the positive and negative gains to correlate with the positive or negative position error. This is particularly useful when controlling hydraulic cylinders in which the different surface areas on each side of the piston react differently with the same gain settings.



The output of the proportional term is proportional to the difference between the commanded position and the actual position read from the feedback device. The primary function of the proportional term is to stabilize the system and speed up the response. It can also be used to reduce the steady state position error.

The SGPN value changes with (tracks) the corresponding positive gain value (SGP) until a SGPN command is executed. After the SGPN command is executed, separate positive and negative gain commands must be used. To re-establish the default mode where SGPN tracks SGP, issue the SGPN command with a minus sign (-) in the command field for the affected axis (e.g., SGPN,- restores axis 2 to the default mode).

When the SGPN command is used alone (i.e., the other gain terms are set to zero), setting this gain too high can cause the system to become oscillatory, underdamped, or even unstable.

NOTE

The SGPN command is specific to the current feedback source (selected with the last SFB command). Therefore, if your application requires switching between feedback sources for the same axis, then for each feedback source, you must select the feedback source with the appropriate SFB command and then issue the SGPN command with the gain values specific to the selected feedback source.

For more information on servo tuning and how the proportional gain affects tuning, refer to your product's *Installation Guide* or to the *Servo Tuner User Guide*.

SGSET Save a Servo Gain Set

		Product	Rev
Type	SERVO	AT6n00	n/a
Syntax	<!>SGSET<i>	AT6n50	1.0
Units	i = gain set identification number	610n	n/a
Range	1 - 5	615n	1.0
Default	n/a	620n	n/a
Response	n/a	625n	1.0
See Also	SFB, SGAF, SGAFN, SGENB, SGI, SGILIM, SGIN, SGP, SGPN, SGV, SGVF, SGVN, SGVFN, SOFFS, SOFFSN, SSWG, TGAIN, TSGSET	6270	1.0

This command allows you to save the presently assigned gain values (SGP, SGI, SGV, SGAF, and SGVF) as a set of gains. Stand-alone servo controllers save (into battery-backed RAM) the gains and the axes and feedback sources to which they are assigned. Up to 5 sets of gains can be saved. Any gain set can be displayed using the TSGSET command.

Any gain set can be enabled with the SGENB command during motion at any specified point in the profile, or when not in motion. For example, you could use one set of gain parameters for the constant velocity portion of the profile, and when you approach the target position a different set of gains can be enabled.

NOTE

The tuning gains in a given gain set are specific to the feedback source that was in use (selected with the last SFB command) at the time the gains were established with the respective gain commands (SGI, SGP, etc.). If your application requires you to switch between feedback sources for the same axis, make sure that the gain set you enable is appropriate to the feedback source you are using at the time.

For more information on servo tuning, refer to your product's *Installation Guide* or to the *Servo Tuner User Guide*.

Example:

```
SGP5,5,10,10    ; Sets the gains for the proportional gain
SGI.1,.1,0,0    ; Sets the gains for the integral gain
SGV50,60,0,0    ; Sets the gains for the velocity gain
SGVF5,6,10,11   ; Sets the gains for the velocity feedforward gain
SGAF0,0,0,0     ; Sets the gains for the acceleration feedforward gain
SGSET3          ; Assign SGP, SGI, SGV, SGVF, & SGAF gains to servo gain set 3
SGP75,75,40,40  ; Sets the gains for the proportional gain
SGI5,5,5,7      ; Sets the gains for the integral gain
SGV1,.45,2,2    ; Sets the gains for the velocity gain
SGVF0,8,0,9     ; Sets the gains for the velocity feedforward gain
SGAF18,20,22,24 ; Sets the gains for the acceleration feedforward gain
SGSET1          ; Assign SGP, SGI, SGV, SGAF, & SGVF gains to servo gain set 1
SGENB1,3,3,1    ; Enables gain set 1 gains on axis 1 &4; enables gain set 3 on
                ; axis 2 & 3
TGAIN           ; Displays the current value for all gains.  Example response:
                ;   *SGP75,5,10,40
                ;   *SGI5,.1,0,7
                ;   *SGV1,60,0,2
                ;   *SGVF0,6,10,9
                ;   *SGAF18,0,0,24
```

SGV — Velocity Feedback Gain

Type	SERVO
Syntax	<!><@><a>SGV<r>,<r>,<r>,<r>
Units	r = microvolts/step/sec
Range	0.00000000 - 2800000.00000000
Default	0
Response	SGV: *SGV0,0,0,0 1SGV: *1SGV0
See Also	ERES, SFB, SGAF, SGI, SGP, SGVF, SGVN, SSWG, TGAIN, TSGSET

Product	Rev
AT6n00	n/a
AT6n50	1.0
610n	n/a
615n	1.0
620n	n/a
625n	1.0
6270	1.0

This command allows you to control the velocity feedback gain in the servo algorithm. Using velocity feedback, the controller's output signal is made proportional to the velocity, or rate of change, of the feedback device position. Since it acts on the rate of change of the position, the action of this term is to anticipate position error and correct it before it becomes too large. This increases damping and tends to make the system more stable.

If this term is too large, the response will be slowed to the point that the system is over-damped. This gain can increase position tracking error, which can be countered by the velocity feedforward term (SGVF).

Since the feedback device signal has finite resolution, the velocity accuracy has a limit. Therefore, if the velocity feedback gain (SGV) is too high, the errors due to the finite resolution are magnified and a noisy, or *chattering*, response may be observed.

NOTE

The SGV command is specific to the current feedback source (selected with the last SFB command). Therefore, if your application requires switching between feedback sources for the same axis, then for each feedback source, you must select the feedback source with the appropriate SFB command and then issue the SGV command with the gain values specific to the selected feedback source.

For more information on servo tuning and how the velocity gain affects tuning, refer to your product's *Installation Guide* or to the *Servo Tuner User Guide*.

Example:

```
SGV100,97,43.334,0  ; Sets the velocity gain term for all the axes
```

SGVF Velocity Feedforward Gain

		Product	Rev
Type	SERVO	AT6n00	n/a
Syntax	<!><@><a>SGVF<r>,<r>,<r>,<r>	AT6n50	1.0
Units	r = microvolts/step/sec	610n	n/a
Range	0.00000000 - 2800000.00000000	615n	1.0
Default	0	620n	n/a
Response	SGVF: *SGVF0,0,0,0 1SGVF: *1SGVF0	625n	1.0
See Also	SFB, SGAF, SGENB, SGI, SGP, SGSET, SGV, SGVFN, SSWG, TGAIN, TSGSET	6270	1.0

Use the Velocity Feedforward Gain (SGVF) command to set the velocity feedforward gain. Introducing velocity feedforward control improves *position tracking performance* when the system is commanded to move at constant velocity. The tracking error is mainly attributed to friction, torque load, and velocity feedback control (SGV).

The SGVF value is multiplied by the *commanded velocity* (calculated by the 6000 controller's DSP move profile routine) to produce the control signal.

Velocity feedforward control can improve the performance of interpolation (linear and circular) application. *However, if your application only requires short, point-to-point moves, velocity feedforward control is not necessary (leave the SGVF command setting at zero—default).*

Because velocity feedforward control is not in the servo feedback loop, it does not affect the servo system's stability, nor does it have any effect at steady state. Therefore, the only limits on how high you can set the velocity feedforward gain (SGVF) are: when it *saturates the control output* (tries to exceed the servo controller's ±10V analog control signal range); or when it causes the actual position to *precede* the commanded position.

NOTE

The SGVF command is specific to the current feedback source (selected with the last SFB command). Therefore, if your application requires switching between feedback sources on the same axis, then for each feedback source, you must select the feedback source with the appropriate SFB command and then issue the SGVF command with the gain values specific to the selected feedback source.

For more information on servo tuning and how the velocity feedforward gain affects tuning, refer to your product's *Installation Guide* or to the *Servo Tuner User Guide*.

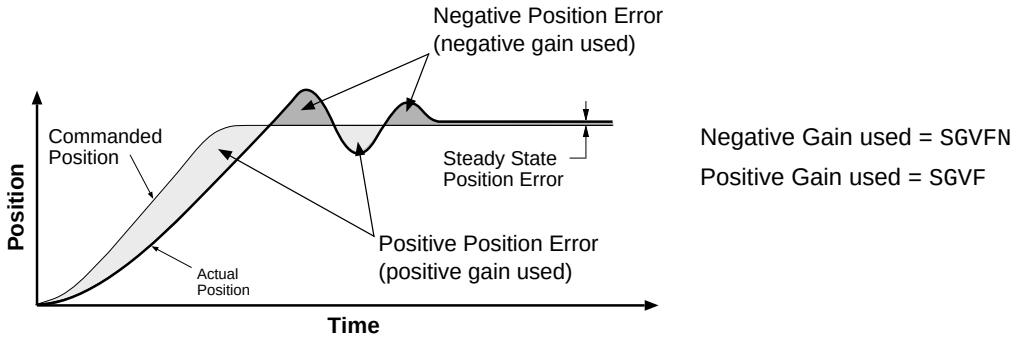
Example:

```
SGVF3555,3555,4000,4000    ; Sets the velocity feedforward for all axes
```

SGVFN — Velocity Feedforward Gain — Negative

		Product	Rev
Type	Servo		
Syntax	<!><@><a>SGVFN<r>,<r>	AT6n00	n/a
Units	r = microvolts/step/sec	AT6n50	n/a
Range	0.00000000 to 2800000.00000000	610n	n/a
Default	0	615n	n/a
Response	SGVFN *SGVFN0,0 1SGVFN *1SGVFN0	620n	n/a
		625n	n/a
See Also	SFB, SGAF, SGAFN, SGENB, SGI, SGIN, SGP, SGPN, SGSET, SGV, SGVF, SGVN, SOFFSN, SSWG, TGAIN, TSGSET	6270	3.0

Use the SGVFN command to set the gain of the velocity feedforward term in the control algorithm when the position error is negative (see illustration below). The controller automatically switches between the positive and negative gains to correlate with the positive or negative position error. This is particularly useful when controlling hydraulic cylinders in which the different surface areas on each side of the piston react differently with the same gain settings.



Introducing velocity feedforward control improves *position tracking performance* when the system is commanded to move at constant velocity. The tracking error is mainly attributed to friction, torque load, and velocity feedback control (SGV and SGVFN). Velocity feedforward control can improve the performance of contouring and linear interpolation applications. *However, if your application only requires point-to-point moves, velocity feedforward control is not necessary (leave the SGVFN command at its default setting).*

The SGVFN value changes with (tracks) the corresponding positive gain value (SGVF) until a SGVFN command is executed. After the SGVFN command is executed, separate positive and negative gain commands must be used. To re-establish the default mode where SGVFN tracks SGVF, issue the SGVFN command with a minus sign (-) in the command field for the affected axis (e.g., SGVFN,- restores axis 2 to the default mode).

The SGVFN value is multiplied by the *commanded velocity* (calculated by the 6000 controller's DSP move profile routine) to produce the control signal. Because velocity feedforward control is not in the servo feedback loop, it does not affect the servo system's stability, nor does it have any effect at steady state. Therefore, the only limits on how high you can set the velocity feedforward gain (SGVF and SGVFN) are: when it *saturates the control output* (tries to exceed the servo controller's ±10V analog control signal range); or when it causes the actual position to *precede* the commanded position.

NOTE

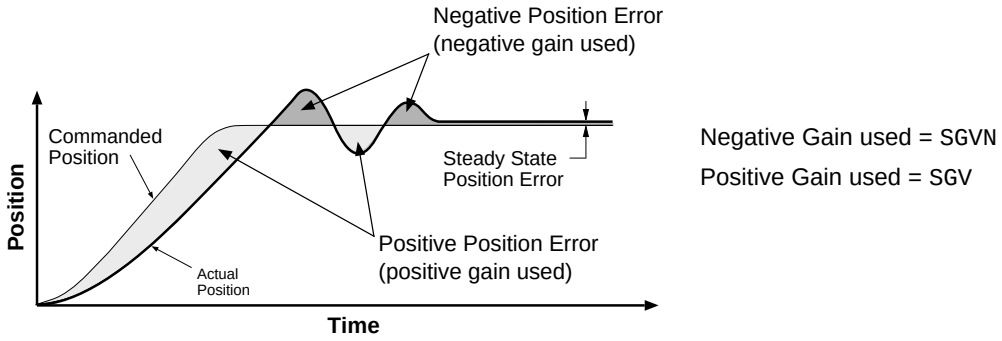
The SGVFN command is specific to the current feedback source (selected with the last SFB command). Therefore, if your application requires switching between feedback sources for the same axis, then for each feedback source, you must select the feedback source with the appropriate SFB command and then issue the SGVFN command with the gain values specific to the selected feedback source.

For more information on servo tuning and how the velocity feedforward gain affects tuning, refer to your product's *Installation Guide* or to the *Servo Tuner User Guide*.

SGVN Velocity Feedback Gain — Negative

		Product	Rev
Type	Servo	AT6n00	n/a
Syntax	<!><@><a>SGVN<r>,<r>	AT6n50	n/a
Units	r = microvolts/step/sec	610n	n/a
Range	0.00000000 to 2800000.00000000	615n	n/a
Default	0	620n	n/a
Response	SGVN *SGVN0,0 1SGVN *1SGVN0	625n	n/a
See Also	SFB, SGAF, SGAFN, SGENB, SGI, SGIN, SGP, SGPN, SGSET, SGV, SGVF, SGVFN, SOFFSN, SSWG, TGAIN, TSGSET	6270	3.0

Use the SGVN command to control the velocity feedback gain in the servo algorithm when the position error is negative (see illustration below). The controller automatically switches between the positive and negative gains to correlate with the positive or negative position error. This is particularly useful when controlling hydraulic cylinders in which the different surface areas on each side of the piston react differently with the same gain settings.



Using velocity feedback, the controller's output signal is made proportional to the velocity, or rate of change, of the feedback device position. Since it acts on the rate of change of the position, the action of this term is to anticipate position error and correct it before it becomes too large. This increases damping and tends to make the system more stable. If this term is too large, the response will be slowed to the point that the system is over-damped.

This gain can increase position tracking error, which can be countered by the velocity feedforward gain (SGVF and SGVFN).

Because the feedback device's signal has a finite resolution, the velocity accuracy has a limit. Therefore, if the velocity feedback gain (SGV) is too high, the errors due to the finite resolution are magnified and a noisy, or *chattering*, response may be observed.

The SGVN value changes with (tracks) the corresponding positive gain value (SGV) until a SGVN command is executed. After the SGVN command is executed, separate positive and negative gain commands must be used. To re-establish the default mode where SGVN tracks SGV, issue the SGVN command with a minus sign (-) in the command field for the affected axis (e.g., SGVN, - restores axis 2 to the default mode).

NOTE

The SGVN command is specific to the current feedback source (selected with the last SFB command). Therefore, if your application requires switching between feedback sources for the same axis, then for each feedback source, you must select the feedback source with the appropriate SFB command and then issue the SGVN command with the gain values specific to the selected feedback source.

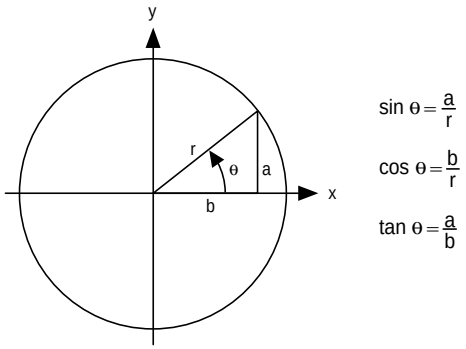
For more information on servo tuning and how the velocity feedback gain affects tuning, refer to your product's *Installation Guide* or to the *Servo Tuner User Guide*.

[SIN( )] Sine

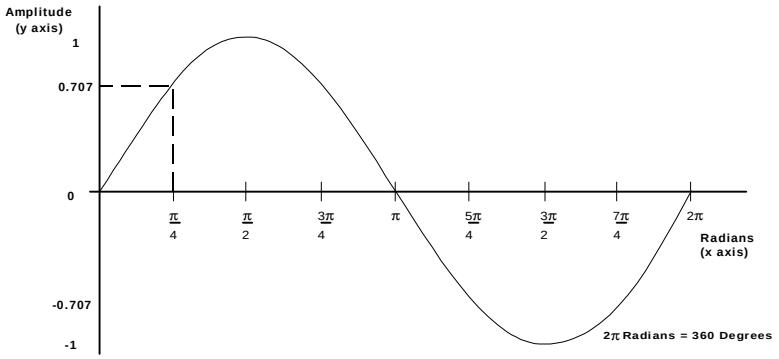
		Product	Rev
Type	Operator (Trigonometric)	AT6n00	1.0
Syntax	... SIN(r) (See below)	AT6n50	1.0
Units	r = value in radian or degrees based on RADIAN command	610n	4.0
Range	0.0000000 to ±17500 radians	615n	1.0
Default	n/a	620n	1.0
Response	n/a	625n	1.0
See Also	[ATAN], [COS], [PI], RADIAN, [TAN], VAR	6270	1.0

This operator is used to calculate the sine of a number given in radians or degrees (see the RADIAN command). If "a" and "b" are coordinates of a point on a circle of radius "r", then the angle of measure "θ" can be defined by the equation: $\sin\theta = \frac{a}{r}$

If a value is given in radians and a conversion is needed to degrees, use the formula: 360° = 2π radians.



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



Syntax: VARx=SIN(r) where x is the numeric variable number and r is a value provided in either degrees or radians based on the RADIAN command. Parentheses (( )) must be placed around the SIN operand. The result will be specified to 5 decimal places.

Example:

```
VAR1=5 * SIN(PI/4)  ; Set variable 1 equal to 5 times the sine of Pi divided by 4
```

SMPER Maximum Allowable Position Error

		Product	Rev
Type	Servo	AT6n00	n/a
Syntax	<!><@><a>SMPER<r>,<r>,<r>,<r>	AT6n50	1.0
Units	r = feedback device steps (scalable with SCLD)	610n	n/a
Range	0 - 200000000 (0 = do not monitor position error condition)	615n	1.0
Default	4000 (1 for 6270, 4096 for 615n)	620n	n/a
Response	SMPER: *SMPER4000,4000,4000,4000 1SMPER: *1SMPER4000	625n	1.0
See Also	[AS], ANIPOL, CMDDIR, ENCPOL, [ER], ERES, ERROR, ERRORP, LDTPOL, SCALE, SCLD, SFB, SGILIM, TANI, TAS, TER, TFB, TLDT, TPC, TPE, TPER	6270	1.0

This command allows you to set the maximum position error allowed before an error condition occurs. The position error, monitored once per system update period, is the difference between the commanded position and the actual position as read by the feedback device selected with the last SFB command. When the position error exceeds the value entered by the SMPER command, an error condition is latched (see TAS or

AS bit #23) and the 6000 controller issues a shutdown to the faulted axis and sets its analog output command to zero volts. To enable the system again, the DRIVE1111 command must be issued, which also sets the commanded position equal to the actual feedback device position (incremental devices will be zeroed).

If the SMPER value is set to zero (SMPERØ), the position error condition is not monitored, allowing the position error to accumulate without causing a fault.

When SMPER is set to a non-zero value, the maximum position error acts as the servo system fault monitor; if the system becomes unstable or loses position feedback, the controller detects the resulting position error, shuts down the drive, and sets an error status bit. You can enable ERROR command bit #12 to continually check for the position error condition, and when it occurs to branch to a programmed response defined in the ERRORP program. You can check the status of this error condition with the TAS, AS, TER, and ER commands. You can check the actual position error with the TPER and PER commands.

If scaling is enabled (SCALE1), the SMPER value is multiplied by the SCLD value.

NOTE

The SMPER command is specific to the current feedback source (selected with the last SFB command). Therefore, if your application requires switching between feedback sources on the same axis, then for each feedback source, you must select the feedback source with the appropriate SFB command and then issue the SMPER command with the gain values specific to the selected feedback source.

Example:

```
ERES4000,4000,4000,4000   ; Set encoder resolution for all axes to 4000 counts/rev
SMPER4000,4000,4000,4000  ; Set maximum allowable position error to 1 rev for
                          ; all axes. If the position error exceeds 4000 counts
                          ; (1 rev) a fault condition will occur.
```

SOFFS Servo Control Signal Offset

Type	Servo
Syntax	<!><@><a>SOFFS<r>,<r>,<r>,<r>
Units	r = volts
Range	-10.000 to 10.000 (resolution is 0.005 volts)
Default	0
Response	SOFFS: *SOFFS0,0,0,0 1SOFFS: *1SOFFS0
See Also	[DAC], DACLIM, DACMIN, SGENB, SGSET, SOFFSN, SSWG, TDAC, TGAIN, TSGSET

Product	Rev
AT6n00	n/a
AT6n50	1.0
610n	n/a
615n	1.0
620n	n/a
625n	1.0
6270	1.0

This command allows you to set an offset voltage to the commanded analog control signal output (commanded analog output + SOFFS value = offset analog output). With this command, you can set an offset voltage to the drive system so that the motor will be stationary in an open-loop configuration. *This is the same effect as the balance input on most analog servo drives.* 6270 Users: If you set the 6270's jumpers for current control, use a voltage-to-current ratio to enter the appropriate offset value in volts.

CAUTION

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

Typically, this offset will be set to zero. This offers a method for setting the analog output command to a known voltage. By setting the SGP, SGI, SGV, SGAF, & SGVF gains to zero, the analog output will reflect this offset value and the system becomes an open-loop configuration.

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

Example:

```
SOFFS0,0,1,2    ; Sets the offset voltage on all axes
```

SOFFSN — Servo Control Signal Offset — Negative

		Product	Rev
Type	Servo	AT6n00	n/a
Syntax	<!><@><a>SOFFSN<r>,<r>	AT6n50	n/a
Units	r = volts	610n	n/a
Range	-10.000 to 10.000 (resolution is 0.005 volts)	615n	n/a
Default	0	620n	n/a
Response	SOFFSN: *SOFFSN0,0,0,0 1SOFFSN: *1SOFFSN0	625n	n/a
See Also	[DAC], DACLIM, SFB, SGAF, SGAFN, SGENB, SGI, SGIN, SGP, SGPN, SGV, SGVF, SGVFN, SGVN, SOFFS, SSWG, TDAC, TGAIN, TSGSET	6270	3.0

This command allows you to set an offset voltage to the commanded analog control signal output when the DAC output is negative. Thus, when the DAC output is negative, the actual analog output comprises the commanded analog output value, plus the SOFFSN value.

The SOFFSN value changes with (tracks) the corresponding positive DAC offset (SOFFS) until a SOFFSN command is executed. After the SOFFSN command is executed, separate positive and negative offset commands must be used. To re-establish the default mode where SOFFSN tracks SOFFS, issue the SOFFSN command with a minus sign (-) in the command field for the affected axis (e.g., SOFFSN,- restores axis 2 to the default mode).

CAUTION — Torque Drive Users

If there is little or no load attached, the SOFFSN offset may cause acceleration to a high speed.

The SOFFS and SOFFSN commands offer a method for setting the analog output command to a known voltage. By setting the SGP/SGPN, SGI/SGIN, SGV/SGVN, SGVF/SGVAFN, & SGVF/SGVFN gains to zero, the system becomes an open-loop configuration. Then you can use the SOFFS and SOFFSN commands to set the analog output command to a known voltage.

Hydraulic Applications (6270):

- The SOFFS and SOFFSN commands can be used to eliminate deadbands due to spool overlap.
- If you set the 6270's jumpers for current control, use a voltage-to-current ratio (___mA range ÷ 10 volts = ___mA/volt) to enter the appropriate offset value in volts. For example, if operating with ±100mA output, the voltage-to-current ratio is 10mA/volt. If you need a 15mA offset, then you would use the SOFFSN1.5 command.

Rotary Applications: With this command, you can set an offset voltage to the drive system so that the load will be stationary in an open-loop configuration. *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 (voltage displayed includes any offset in effect).

[SQRT()] — Square Root

		Product	Rev
Type	Operator (Mathematical)	AT6n00	1.0
Syntax	See below	AT6n50	1.0
Units	n/a	610n	4.0
Range	n/a	615n	1.0
Default	n/a	620n	1.0
Response	n/a	625n	1.0
See Also	[=], [+], [-], [*], [/], VAR	6270	1.0

This operator takes the square root of a value. The result, if multiplied by itself, will *approximately equal* the original value (the difference is attributed to round-off error). The resulting value has 3 decimal places.

Syntax: VARn=SQRT(expression) where n is the variable number, and the expression can be a number or a mathematical expression. The SQRT of a negative number is not allowed. Parentheses (( )) must be placed around the SQRT operand.

Example:

```
VAR1=SQRT(25)   ; Set variable 1 equal to the square root of 25 (result = 5)
```

[SS] System Status

Type	Assignment or Comparison
Syntax	See below
Units	n/a
Range	n/a
Default	n/a
Response	n/a
See Also	IF, TCMDER, TRGFN, TSS, TSSF, TSTAT, VARB

Product	Rev
AT6n00	2.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	2.0
625n	1.0
6270	1.0

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

Syntax: VARBn=SS where n is the binary variable number, or [SS] can be used in an expression such as IF(SS=b11Ø1), or IF(SS=h7F)

The function of each system status bit is shown below.

BIT (Left to Right)	**Function** (1 = yes, Ø = no)	**BIT** (Left to Right)	**Function** (1 = yes, Ø = no)
1	System Ready	17	Loading Thumbwheel Data ([TW])
2	Reserved	18	External Program Select Mode (INSELP)
3	Executing a Program	19	Dwell in Progress (T command)
4	Immediate Command (set if last command was immediate)	20	Waiting for RP240 Data—[DREAD] or [DREADF] (stand-alone products only)
5	In ASCII Mode	21	RP240 Connected (stand-alone products only) — current PORT setting only
6	In Echo Mode (stand-alone products only) — current PORT setting only	22	Non-volatile Memory Error (stand-alone products only)
7	Defining a Program	23	Servo data gathering transmission in progress (servo products only)
8	In Trace Mode	24	Reserved
9	In Step Mode	25 *	Position captured with **TRG-A**
10	In Translation Mode (bus-based products must use fast status area to see)	26 *	Position captured with **TRG-B**
11	Command Error Occurred (bit is cleared when TCMDER is issued)	27 *	Position captured with **TRG-C**
12	Break Point Active (BP)	28 *	Position captured with **TRG-D**
13	Pause Active	29	Compiled memory is 75% full
14	Wait Active (WAIT)	30	Compiled memory is 100% full
15	Monitoring On Condition (ONCOND)	31 **	Compile operation failed (PCOMP) **
16	Waiting for Data (READ)	32	Reserved

* Bits 25 through 28 are cleared when the captured position is read with the [CA], [PCA], [PCC], [PCE], [PCL], [PCM], TCA, TPCA, TPCC, TPCE, TPCL, or TPCM commands, but the position information is still available from the respective registers until it is overwritten by a subsequent position capture.

** Bit #31: failed PCOMP compile is cleared on power up, RESET, or after successful compile. Possible causes include:

- Errors in profile design (e.g., change direction while at non-zero velocity; distance & velocity equate to < 1 count per system update; preset move profile ends in non-zero velocity)
- Profile will cause a Following error (see TFSF, TFS, or FS command descriptions)
- Out of memory (see SS bit #30)
- Axis already in motion at the time of the PCOMP command
- Loop programming errors (e.g., no matching PLOOP or PLN; more than 4 embedded PLOOP/END loops)

If it is desired to assign only one bit of the system status value to a binary variable, instead of all 32, the bit select (.) operator can be used. For example, VARB1=SS.12 assigns system status bit 12 to binary variable 1: *VARB1=XXXX_XXXX_XXXØ_XXXX_XXXX_XXXX_XXXX_XXXX.

Example:

```
VARB1=SS           ; System status assigned to binary variable 1
IF(SS=b111011X11)  ; If the system status contains 1s in bit locations 1, 2, 3,
                   ; 5, 6, 8, & 9, and a 0 in bit location 4, do the IF
                   ; statement
IF(SS=h7F00)       ; If the system status contains 1s in bit locations 1, 2, 3,
                   ; 5, 6, 7, & 8, and 0s in every other bit location, do the IF
                   ; statement
NIF                ; End of second IF statement
NIF                ; End of first IF statement
```

SSFR — Servo Sampling Frequency Ratio

Type	Servo
Syntax	<!>SSFR<i>
Units	i = sampling ratio number
Range	1, 2, 4, or 8
Default	4
Response	SSFR: *SSFR4
See Also	ERES, INDAX, INDEB, INFNC, LDTUPD, SDTAMP, SDTFR

Product	Rev
AT6n00	n/a
AT6n50	3.0
610n	n/a
615n	3.0
620n	n/a
625n	3.0
6270	3.0

NOTE
SSFR will not be effective unless the drive(s) are disabled (@DRIVEØ)

A coarse commanded position is computed and updated at the *motion trajectory update rate*. This course commanded position is interpolated at the *servo sampling update rate* to produce a smoother continuous commanded position. The servo control signal computed by the servo algorithm is also updated at the servo sampling update rate. The ratio between these two update rates is determined by the Servo Sampling Frequency Ratio (SSFR) command, which offers four selectable ratio settings. These four ratios and the actual sampling frequencies and sampling periods (reciprocal of sampling frequency) are shown in the table below.

The ratio between the motion trajectory and servo sampling update rates has a direct effect on the *system update rate*. The system update rate is the rate for I/O updates, input debounce, timer resolution, fast status update (bus-based controllers), and LDT position update (6270).

# of Axes Active (INDAX)	SSFR Command Setting	Servo Sampling Update Frequency (samples/sec.)	Servo Sampling Update Period (µsec)	Motion Trajectory Update Frequency (samples/sec.)	Motion Trajectory Update Period (µsec)	System Update Frequency (samples/sec.)	System Update Period (µsec)
INDAX1	SSFR1	3030	330	3030	330	757	1320
INDAX1	SSFR2	5405	185	2703	370	675	1480
* INDAX1	SSFR4	6250	160	1563	640	520	1920
INDAX1	SSFR8	6667	150	833	1200	417	2400
INDAX2	SSFR1	2353	425	2352	425	588	1700
INDAX2	SSFR2	3571	280	1786	560	446	2400
** INDAX2	SSFR4	3571	280	893	1120	446	2400
INDAX2	SSFR8	3571	280	446	2240	446	2400
INDAX3	SSFR1	1667	600	1667	600	555	1800
INDAX3	SSFR2	2222	450	1111	900	555	1800
INDAX3	SSFR4	2353	425	588	1700	588	1700
INDAX4	SSFR1	1250	800	1250	800	417	2400
INDAX4	SSFR2	1667	600	833	1200	417	2400
*** INDAX4	SSFR4	2000	500	500	2000	500	2000

* Factory default settings for single-axis controllers
** Factory default settings for two-axis controllers
***Factory default settings for four-axis controllers

The general rule to determining the proper SSFR value is to first select the slowest servo sampling frequency that is able to give a satisfactory response. This can be done by experiment or based on the closed-loop bandwidth requirement for your application. (**NOTE**: Increasing the SSFR value allows for higher bandwidths, but produces a rougher motion profile; conversely, decreasing the SSFR value provides a smoother profile, but makes the servo system less stable and slower to respond.)

As an example, let's say your application requires a closed-loop bandwidth of 120 Hz. If you determine the minimum servo sampling frequency by using the rule of thumb—setting the servo sampling frequency at least 8 times higher than the bandwidth frequency—the required minimum servo sampling frequency would be 1000 Hz. If four axes are running (INDAX4), then you should try using the SSFR1 setting.

The following table provides a general guideline for various application requirements.

Application Requirement	SSFR1	SSFR2	SSFR4	SSFR8
X-Y Linear interpolation	x	x		
Fast point-to-point motion			x	x
Regulation (speed, torque, etc.)			x	x
High natural frequency system				x

Example:
```
SSFR4           ; Sets the ratio of commanded position updates to servo
                ; control updates to 4
```

SSV — Start/Stop Velocity

		Product	Rev
Type	Motion		
Syntax	<!><@><a>SSV<r>,<r>,<r>,<r>	AT6n00	1.0
Units	r = units/sec	AT6n50	n/a
Range	0.00000 - 1,600,000 (depends on scale factor and PULSE)	610n	4.0
Default	0.0000	615n	n/a
Response	SSV: *SSV0.0000,0.0000,0.0000,0.0000 1SSV: *1SSV0.0000	620n	1.0
		625n	n/a
See Also	GO, PULSE, SCALE, SCLV, V	6270	n/a

The Start/Stop Velocity (SSV) command specifies the instantaneous velocity to be used when starting or stopping. By using the SSV command, there will be no acceleration from Ø units/sec to the SSV value, instead motion will immediately begin with a velocity equal to the SSV value.

This command is useful for accelerating past low-speed resonant points, where a full- or half-stepping drive may stall. *With microstepping systems, this command is not necessary.*

If scaling is not enabled (SCALEØ), the velocity value is entered in motor revs/sec; this value is internally multiplied by the drive resolution (DRES) value to obtain a velocity value in motor steps/sec for the motor trajectory calculations. If scaling is enabled (SCALE1), the entered velocity value is internally multiplied by the velocity scaling factor (SCLV) to convert user units/sec to motor steps/sec. The entered value is always in reference to motor steps, not encoder steps, regardless of the ENC command setting.

Scaling: If scaling is enabled, the SSV command value entered is internally multiplied by the velocity scaling factor (SCLV). The velocity value may be truncated if the value entered exceeds the velocity resolution at the given scaling factor. Refer to the SCLV command description for additional information on velocity scaling.

Example:
```
@SSV1           ; Set start/stop velocity to 1 unit/sec on all axes
A200,200,1,1    ; Set acceleration to 200 units/sec/sec for axes 1 & 2,
                ; and 1 unit/sec/sec for axes 3 & 4
AD400,400,1,1   ; Set deceleration to 400 units/sec/sec for axes 1 & 2,
                ; and 1 unit/sec/sec for axes 3 & 4
V10,10,10,20    ; Set velocity to 10, 10, 10, & 20 units/sec for axes 1, 2, 3 & 4
@D1000          ; Set distance on all axes to 1000 units
GO1100          ; Initiate motion on axes 1 & 2. The motors will start at a
                ; velocity of 1 unit/sec and accelerate up to 10 units/sec,
                ; travel at 10 units/sec, and then decelerate down from
                ; 10 units/sec to 1 unit/sec where they will
                ; instantaneously stop.
```

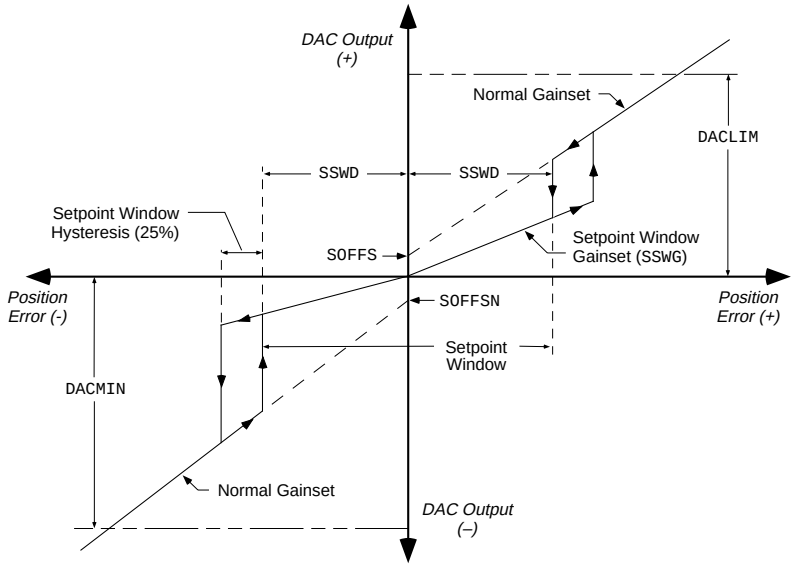
SSWD Setpoint Window Distance

		Product	Rev
Type	Servo		
Syntax	<!><@><a>SSWD<r>,<r>	AT6n00	n/a
Units	r = distance units (scalable with SCLD)	AT6n50	n/a
Range	0 to 999999999	610n	n/a
Default	0	615n	n/a
Response	SSWD: *SSWD0,0 1SSWD: *1SSWD0	620n	n/a
		625n	n/a
See Also	SCLD, SGSET, SSWG, STRGTE	6270	3.0

The SSWD command defines the distance on both sides of the position setpoint ("setpoint window") in which the gain set specified with the SSWG command is used. Specifically, the gain set is automatically invoked by the controller *after* the commanded move profile is complete and the actual position is within the setpoint window. The setpoint window includes a hysteresis loop equal to 25% of the value used in the SSWD command.

To assign a gain set as the setpoint window gain set with the SSWG command, you must first define/save the gain set with the SGSET command (see programming example below).

The diagram at right makes two assumptions. First, for simplicity, only a proportional gain is being used. Second, in the SSWG gain set the proportional gains (SGP and SGPN) are lower than those used in the normal gains and the offsets (SOFFS and SOFFSN) are set to zero.



HINT: You can use the target zone settling mode to override the setpoint window distance (SSWD) and introduce distance and velocity end-of-move criteria to define when the controller switches to the setpoint window gains. When using the target zone settling mode (enabled with the STRGTE command), after completion of the commanded move profile, the actual position and actual velocity must be within the "target zone" (i.e., within the position band defined by STRGTD and within the velocity band defined by STRGTV) before motion can be determined complete. After that point, the controller will switch to the SSWG gains, even if a target zone timeout occurs (STRGTT value exceeded). There is no hysteresis when using the target zone settling mode.

Example:

```
SGP35     ; Set proportional gain to 35
SGI3      ; Set integral gain to 3
SGSET3    ; Save current gains as gain set #3 (for use later in the
          ; setpoint window)
SSWG3     ; Assign gain set #3 as the setpoint window gain set
SGP50     ; While moving, use higher proportional gain, no integral gain,
          ; but introduce velocity gain
SGI0
SGV4
SSWD100   ; After the commanded move profile is complete, the controller
          ; will use gain set #3 if within 100 counts (125 counts including
          ; hysteresis) of the setpoint position
```

SSWG Setpoint Window Gain Set

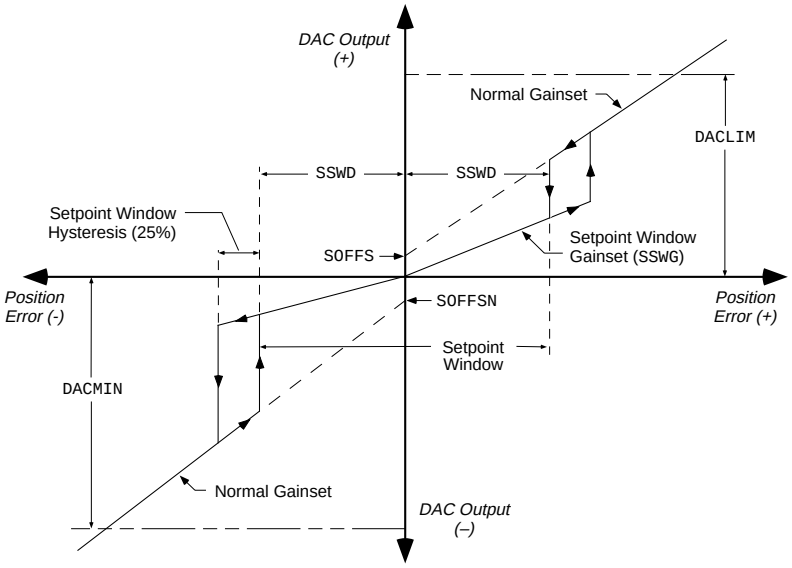
		Product	Rev
Type	Servo	AT6n00	n/a
Syntax	<!><@><a>SSWG<i>,<i>	AT6n50	n/a
Units	i = gain set identification number	610n	n/a
Range	1-5	615n	n/a
Default	0 (feature disabled—use normal gain set)	620n	n/a
Response	SSWG: *SSWG0,0 1SSWG: *1SSWG0	625n	n/a
See Also	SFB, SGAF, SGAFN, SGENB, SGI, SGIN, SGP, SGPN, SGSET, SGV, SGVF, SGVFN, SGVN, SOFFS, SOFFSN, SSWD, STRGTE, TGAIN, TSGSET	6270	3.0

The SSWG command assigns a pre-defined gain set to automatically be used *after* the commanded move profile is complete and the actual position is within the setpoint window established with the SSWD command. *To disable the setpoint window gain feature, use the SSWGØ command (automatically disabled if SFBØ).*

To assign a gain set as the setpoint window gain set with the SSWG command, you must first define/save the gain set with the SGSET command (see programming example below). When you issue the SGSET command, all of the items listed below are saved to the specified gain set. To ascertain the values of the current active gains, use the TGAIN transfer command. To find out what gain values are used in a particular gain set, use the TSGSET transfer command.

SGP and SGPN	Proportional gains (positive and negative)
SGV and SGVN	Velocity gains (positive and negative)
SGI and SGIN	Integral gains (positive and negative)
SGVF and SGVFN	Velocity feedforward gains (positive and negative)
SGAF and SGAFN	Acceleration feedforward gains (positive and negative)
SGILIM	Integral Windup Limit
SOFFS and SOFFSN	Servo Control Signal Offset (positive and negative)

The diagram at right makes two assumptions. First, for simplicity, only a proportional gain is being used. Second, in the SSWG gain set the proportional gains (SGP and SGPN) are lower than those used in the normal gains and the offsets (SOFFS and SOFFSN) are set to zero.



After completion of the commanded move profile, the SSWG gains are automatically used when the position error is within the "setpoint window". The SSWG gain set accommodates different proportional, integral, and velocity gains and offsets for each direction. If you want to disable the servo control loop when the position error is within the window, set all of the gain values to zero (you can do this by executing the SFBØ command). If you want to disable the offsets (SOFFS & SOFFSN) when the position error is within the setpoint window, set them to zero volts.

The arrows on the above diagram illustrate the hysteresis loop. The SSWG gains are used until the position error increases to a value of 25% greater than the number specified in the SSWD command. At this point, the normal gain set is automatically substituted until the position error falls below the value in the SSWD command when the SSWG gains are returned.

Finally, the maximum positive DAC voltage is determined by the value in the DACLIM command and the maximum negative value is determined by the DACMIN command.

HINT: You can use the target zone settling mode to override the setpoint window distance (SSWD) and introduce distance and velocity end-of-move criteria to define when the controller switches to the setpoint window gains. When using the target zone settling mode (enabled with the STRGTE command), after completion of the commanded move profile, the actual position and actual velocity must be within the "target zone" (i.e., within the position band defined by STRGTD and within the velocity band defined by STRGTV) before motion can be determined complete. After that point, the controller will switch to the SSWG gains, even if a target zone timeout occurs (STRGTT value exceeded). There is no hysteresis when using the target zone settling mode.

Example:

```
SGP35      ; Set proportional gain to 35
SGI3       ; Set integral gain to 3
SGSET3     ; Save current gains as gain set #3 (for use later in the
           ; setpoint window)
SSWG3      ; Assign gain set #3 as the setpoint window gain set
SGP50      ; While moving, use higher proportional gain, no integral gain,
           ; but introduce velocity gain
SGI0
SGV4
SSWD100    ; After the commanded move profile is complete, the controller
           ; will use gain set #3 if within 100 counts (125 counts including
           ; hysteresis) of the setpoint position
```

STARTP Start-Up Program

Type	Subroutines
Syntax	<!>STARTP<t>
Units	t = text (name of program)
Range	Text name of 6 characters or less
Default	n/a
Response	STARTP: *STARTP MAIN
See Also	DEF, RESET, SCALE

Product	Rev
AT6n00	n/a
AT6n50	n/a
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Start-Up Program (STARTP) command specifies the name of the program that will automatically be run upon power-up and RESET. If the program that is identified as the STARTP program is deleted with the DEL command, the STARTP is automatically cleared. If you wish to prevent the STARTP program from being executed, without having to delete the assigned program, issue the STARTP CLR command.

This command applies only to stand-alone 6000 series products, not bus-based products.

Example:

```
STARTP WakeUp   ; Set program WakeUp as the program that will start to run
                ; after power is cycled or the 6000 product is reset
STARTP CLR      ; Clears the program WakeUp from its assignment as the
                ; start-up program
DEL WakeUp      ; Deletes the program WakeUp and clears the STARTP command
                ; (no power-up program will be executed)
```

STD Streaming Interval

Type	Motion
Syntax	<!><@>STD<i>
Units	i = milliseconds
Range	10 - 50 (only in even numbers)
Default	10
Response	STD: *STD10
See Also	SD, STREAM

Product	Rev
AT6n00	1.1
AT6n50	n/a
610n	n/a
615n	n/a
620n	n/a
625n	n/a
6270	n/a

The Streaming Interval (STD) command sets the time interval for execution of Streaming Data (SD) commands. If the STREAM command is set to 1, then for each STD interval, the motor travels the number of

motor steps set by the SD command. With the STREAM command set to 2, the motor will travel at the velocity set by the SD command during the STD interval.

Example:

```
DEF SAMPLE      ; Begin definition of program named sample
PULSE1          ; Set pulse width to 1 ms
STD20           ; Set streaming interval to 20 milliseconds (ms)
STREAM1         ; Set distance streaming mode
SD12            ; Travel 12 steps positive-direction in 20 ms
SD25            ; Travel 25 steps positive-direction in 20 ms
SD50            ; Travel 50 steps positive-direction in 20 ms
SD700000000     ; Exit streaming mode
WAIT(MOV=b0)    ; Wait for motion to stop
END             ; End program definition
SAMPLE          ; Initiate program sample
```

STEP — Single Step Mode Enable

Type	Program Debug Tool
Syntax	<!>STEP<b>
Units	n/a
Range	b = 0 (disable), 1 (enable) or X (don't care)
Default	0
Response	STEP: *STEP0
See Also	[#], BP, [SS], TRACE, TRANS, TSS

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Single Step Mode Enable (STEP) command enables single command step mode. Single step mode is used for stepping through a defined (DEF) program. To execute single step mode:

1. Define a program (DEF)
2. Enable single step mode (STEP1)
3. Run the program (RUN)
4. Use the immediate pound (!#) to step through the program

Each step (!#) command will initiate the next command to be processed.

Example:

```
DEF tester      ; Begin definition of program named tester
V1,1,1,1        ; Set velocity to 1 unit/sec on all axes
A10,10,10,10    ; Set acceleration to 10 units/sec/sec on all axes
                ; (Note: This command will not be executed until a !# sign
                ; is received.)
D1,2,3,4        ; Set distance to 1 unit on axis 1, 2 units on axis 2,
                ; 3 units on axis 3, and 4 units on axis 4
GO1101          ; Initiate motion on axes 1, 2, and 4
OUT11X1         ; Turn on programmable outputs 1, 2, and 4, leave 3 unchanged
END             ; End program definition
STEP1           ; Enable single step mode
RUN tester      ; Execute program named tester
; ************************************************************************
; * At this point no action will occur because single step mode          *
; * has been enabled.  Here's how to execute commands:                   *
; *  !#2  (Execute 1st 2 commands in the program: V1,1,1,1 & A10,10,10,10) *
; *  !#   (Execute 1 command: command to be executed is D1,2,3,4)         *
; *  !#1  (Execute 1 command: command to be executed is GO110)           *
; *  !#2  (Execute 2 commands: commands to be executed are OUT11X1 & END)  *
; ************************************************************************
```

STREAM Streaming Mode

Type	Motion
Syntax	<!><@><a>STREAM<i>,<i>,<i>,<i>
Units	n/a
Range	i = 0 (exit), 1 (distance streaming), or 2 (velocity streaming)
Default	0
Response	STREAM: *STREAM0,0,0,0
See Also	[AS], PULSE, SD, STD, TAS

Product	Rev
AT6n00	1.1
AT6n50	n/a
610n	n/a
615n	n/a
620n	n/a
625n	n/a
6270	n/a

The Streaming Mode (STREAM) command sets the indexer to a streaming configuration. A value of 1 (STREAM1) enables the Distance Streaming mode, where data in the Streaming Data (SD) command represents a motor step distance. A value of 2 (STREAM2) enables the Velocity Streaming Mode, where data in the SD command indicates velocity values. Entering Ø (STREAMØ) for any axis will exit the streaming mode for **all** axes. The SD data is executed once per a time interval set by the STD command. All the streaming axes must enter the streaming mode with the same STREAM command.

While in the streaming mode, the SD commands (and any other commands present) are executed *on-the-fly* (like the Continuous Command Execution Mode), regardless of the COMEXC command setting. Actual processing of SD commands begins after ten SD commands (maximum of four datapoints per SD command) have been processed or an Exit Streaming Mode (SD7ØØØØØØØØØ) command is encountered. The moving/not moving bit in the axis status register is set after ten SD commands have been processed, and remains active during the entire streaming process.

CAUTION

Placing commands other than SD commands in a program may cause mispositioning if the command takes too long to execute. Status should be monitored via the fast status area.

A Pause (PS), Kill (K) or Stop (S) command will exit the streaming mode. Encountering a hardware or software limit will also exit the streaming mode. No deceleration will be performed.

NOTE

To enter the Streaming Mode, you must set the PULSE command to 1 µs or greater.

Example:

```
DEF SAMPLE      ; Begin definition of program sample
PULSE1          ; Set pulse width to 1 ms
STD20           ; Set streaming interval to 20 ms
STREAM2         ; Set velocity streaming mode
SD12            ; Run at velocity value 12 for 20 ms
SD25            ; Run at velocity value 25 for 20 ms
SD36            ; Run at velocity value 36 for 20 ms
SD700000000     ; Exit streaming mode
WAIT(MOV=b0)    ; Wait for motion to stop
END             ; End definition of program sample
SAMPLE          ; Execute program sample
```

STRGTD Target Distance Zone

Type	Servo
Syntax	<!><@><a>STRGTD<r>,<r>,<r>,<r>
Units	r = distance units (scalable)
Range	0 - 999999999
Default	50
Response	STRGTD: *STRGTD50,50,50,50 1STRGTD: *1STRGTD50
See Also	[AS], SCLD, SSWD, SSWG, STRGTE, STRGTT, STRGTV, TAS, TSTLT

Product	Rev
AT6n00	4.0
OEM-AT6n00	n/a
AT6n50	1.0
610n	4.0
615n	1.0
620n	4.0
625n	1.0
6270	1.0

This command sets the target distance zone used in the Target Zone Settling Mode. The target distance zone is a range of positions around the desired endpoint that the load must be within before motion is considered complete. If scaling is enabled (SCALE1), the STRGTD value is multiplied by the distance scale factor (SCLD).

When using the Target Zone Mode, the load's actual position and actual velocity must be within the *target zone* (that is, within the distance zone defined by STRGTD and within the velocity zone defined by STRGTV) before motion can be determined complete.

If the load does not settle into the target zone before the timeout period set by STRGTT, the controller detects an error (see TAS or AS bit #25). If this error occurs, you can prevent subsequent command and/or move execution by enabling the ERROR command to continually check for this error condition, and when it occurs to branch to a programmed response defined in the ERRORP program. (Refer to the ERRORP command description for an example of using an error program.)

*** *For a more information on target zone operation, refer to the 6000 Series Programmer's Guide.*

Example:

```
STRGTD5,5,5,5          ; Sets the distance target zone to +/-5 units
STRGTV.01,.01,.01,.01  ; Sets the velocity target zone to <= 0.01 units/sec
STRGTT10,10,10,10      ; Sets the timeout period to 10 milliseconds on all axes
STRGTE1111             ; Enables the target zone criterion for all axes
;
; Given these target zone commands, a move with a distance of 8,000 units
; (@D8000) must end up between position 7,995 and 8,005 and settle down
; to <=0.01 units/sec within 10 ms after the commanded profile is complete.
```

STRGTE Enable Target Zone Settling Mode

Type	Servo
Syntax	<!><@><a>STRGTE<b><b><b><b>
Units	n/a
Range	b = 0 (disable), 1 (enable), or X (don't care)
Default	0
Response	STRGTE: *STRGTE0011 1STRGTE: *1STRGTE0
See Also	COMEXC, ENC, SSWG, STRGTD, STRGTT, STRGTV, TSTLT

Product	Rev
AT6n00	4.0
OEM-AT6n00	n/a
AT6n50	1.0
610n	4.0
615n	1.0
620n	4.0
625n	1.0
6270	1.0

This command enables or disables the Target Zone Settling Mode. When using the target zone settling criterion, the load's actual position and actual velocity must be within the *target zone* (that is, within the position band defined by STRGTD and within the velocity band defined by STRGTV) before motion can be determined complete. **STEPPER PRODUCTS** must be in encoder step mode (ENC1) to use this feature.

If the load does not settle into the target zone before the timeout period set by STRGTT, the controller detects an error (see TAS or AS bit #25). If this error occurs, you can prevent subsequent command and/or move execution by enabling the ERROR command to continually check for this error condition, and when it occurs to branch to a programmed response defined in the ERRORP program.

*** *For a more information on target zone operation, refer to the 6000 Series Programmer's Guide.*

Example:

```
STRGTD5,5,5,5          ; Sets the distance target zone to +/-5 units
STRGTV.01,.01,.01,.01  ; Sets the velocity target zone to <= 0.01 units/sec
STRGTT10,10,10,10      ; Sets the timeout period to 10 milliseconds on all axes
STRGTE1111             ; Enables the target zone criterion for all axes
;
; Given these target zone commands, a move with a distance of 8,000 units
; (@D8000) must end up between position 7,995 and 8,005 and settle down
; to <=0.01 units/sec within 10 ms after the commanded profile is complete.
```

STRGTT Target Settling Timeout Period

Type	Servo
Syntax	<!><@><a>STRGTT<i>,<i>,<i>,<i>
Units	r = milliseconds
Range	0 - 5000
Default	1000
Response	STRGTT: *STRGTT1000,1000,1000,1000 1STRGTT: *1STRGTT1000
See Also	[AS], [ER], ERROR, ERRORP, SSWG, STRGTD, STRGTE, STRGTV, TAS, TER, TSTLT

Product	Rev
AT6n00	4.0
OEM-AT6n00	n/a
AT6n50	1.0
610n	4.0
615n	1.0
620n	4.0
625n	1.0
6270	1.0

This command sets the maximum time allowed for the load to settle within the defined target zone before an error occurs.

This command is useful only if the Target Zone Settling Mode is enabled with the STRGTE command. When using the Target Zone Settling Mode, the load's actual position and actual velocity must be within the *target zone* (that is, within the position band defined by STRGTD and within the velocity zone defined by STRGTV) before motion can be determined complete. If the load does not settle into the target zone before the timeout period set by STRGTT, the servo controller detects an error (see TAS or AS bit #25).

If this error occurs, you can prevent subsequent command and/or move execution by enabling the ERROR command to continually check for this error condition, and when it occurs to branch to a programmed response defined in the ERRORP program. (Refer to the ERRORP command description for an example of using an error program.) You can check the status of the error condition with the TER and ER commands.

**** For a more information on target zone operation, refer to the 6000 Series Programmer's Guide.*

Example:

```
STRGTD5,5,5,5          ; Sets the distance target zone to +/-5 units
STRGTV.01,.01,.01,.01  ; Sets the velocity target zone to <= 0.01 units/sec
STRGTT10,10,10,10      ; Sets the timeout period to 10 milliseconds on all axes
STRGTE1111             ; Enables the target zone criterion for all axes
;
; Given these target zone commands, a move with a distance of 8,000 units
; (@D8000) must end up between position 7,995 and 8,005 and settle down
; to <=0.01 units/sec within 10 ms after the commanded profile is complete.
```

STRGTV Target Velocity Zone

Type	Servo
Syntax	<!><@><a>STRGTV<r>,<r>,<r>,<r>
Units	r = units/sec (scalable by SCLV)
Range	0 - 200 rps
Default	1.0000
Response	STRGTV: *STRGTV1.0000,1.0000,0,0 1STRGTV: *1STRGTV1.0000
See Also	[AS], SCLV, SSWG, STRGTD, STRGTE, STRGTT, TAS, TSTLT

Product	Rev
AT6n00	4.0
OEM-AT6n00	n/a
AT6n50	1.0
610n	4.0
615n	1.0
620n	4.0
625n	1.0
6270	1.0

This command sets the target velocity zone for use in the Target Zone Settling Mode. The target velocity zone is a velocity range that the load must be within before motion is considered complete. If scaling (SCALE) is enabled, the STRGTV value is multiplied by the velocity scale factor (SCLV).

When using the Target Zone Mode, the load's actual position and actual velocity must be within the *target zone* (that is, within the distance zone defined by STRGTD and less than or equal to the velocity defined by STRGTV) before motion can be determined complete.

If the load does not settle into the target zone before the timeout period set by STRGTT, the servo controller detects an error (see TAS or AS bit #25). If this error occurs, you can prevent subsequent command and/or move execution by enabling the ERROR command to continually check for this error condition, and when it occurs to branch to a programmed response defined in the ERRORP program. (Refer to the ERRORP command description for an example of using an error program.)

*** *For a more information on target zone operation, refer to the 6000 Series Programmer's Guide.*

Example:

```
STRGTD5,5,5,5          ; Sets the distance target zone to +/-5 units
STRGTV.01,.01,.01,.01  ; Sets the velocity target zone to <= 0.01 units/sec
STRGTT10,10,10,10      ; Sets the timeout period to 10 milliseconds on all axes
STRGTE1111             ; Enables the target zone criterion for all axes
;
; Given these target zone commands, a move with a distance of 8,000 units
; (@D8000) must end up between position 7,995 and 8,005 and settle down
; to <=0.01 units/sec within 10 ms after the commanded profile is complete.
```

T — Time Delay

		Product	Rev
Type	Program Flow Control	AT6n00	1.0
Syntax	<!>T<r>	AT6n50	1.0
Units	r = seconds	610n	4.0
Range	0.001 - 999.999	615n	1.0
Default	n/a	620n	1.0
Response	n/a	625n	1.0
See Also	GOWHEN, PS, [SS], SSFR, [TIM], TTIM, TSS, WAIT	6270	1.0

The Time Delay (T) command pauses command processing for **r** seconds before continuing command execution. Once the elapsed time has expired, the command after the T command will be executed.

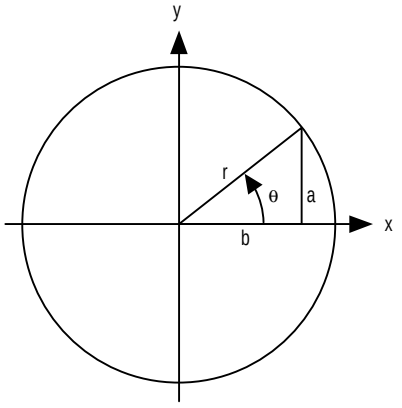
The minimum resolution of the T command is: 2 ms for the stepper products, and 1 *system update period* for the servo products (see table in SSFR command description). Although you can enter time delays that are not multiples of 2 ms, the time delay will be rounded up to the next multiple of 2 ms. For example, T.005 produces a 6 ms time delay in the stepper products.

Example:
```
T5              ; Wait 5 seconds before executing TPE command
TPE             ; Transfer position of all encoders to the terminal
```

[TAN( )] — Tangent

		Product	Rev
Type	Operator (Trigonometric)	AT6n00	1.0
Syntax	... TAN(r) (See below)	AT6n50	1.0
Units	r = radians or degrees depending on RADIAN command	610n	4.0
Range	0.0000000 to ±17500 radians	615n	1.0
Default	n/a	620n	1.0
Response	n/a	625n	1.0
See Also	[ATAN], [COS], [PI], RADIAN, [SIN], VAR	6270	1.0

The Tangent (TAN) operator is used to calculate the tangent of a number given in radians or degrees (see the RADIAN command). If "a" and "b" are coordinates of a point on a circle of radius "r", then the angle of measure "θ" can be defined by the equation: $\tan\theta = \frac{a}{b}$



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

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

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

If a value is given in radians and a conversion is needed to degrees, use the following formula: 360° = 2π radians.

Syntax: VARx=TAN(r), where x is the numeric variable number and r is a value in either radians or degrees depending on the RADIAN command. Parentheses (( )) must be placed around the TAN operand. The result will be specified to 5 decimal places.

Example:
```
VAR1=5 * TAN(PI/4)   ; Set variable 1 = 5 times the tangent of Pi divided by 4
```

TANI — Transfer Analog Input Voltage (-ANI Option Board)

Type	Transfer
Syntax	<!><i>TANI
Units	i = analog input identifier
Range	1 - 4 for AT6n50; 1 - 2 for 625n & 6270; 1 for 615n
Default	n/a
Response	TANI: *TANI1.963,1.453 1TANI: *1TANI1.963
See Also	[ANI], [CA], [FB], [PANI], [PCA], TFB, TPANI, TPCA

Product	Rev
AT6n00	n/a
AT6n50-ANI	1.0
610n	n/a
615n-ANI	1.0
620n	n/a
625n-ANI	1.0
6270-ANI	1.0

The Transfer Analog Input Voltage for the -ANI option (TANI) command returns the voltage level present at the ANI analog inputs. The value reported with the TANI command is measured in volts and does not reflect the effects of distance scaling (SCLD), position offset (PSET), polarity reversal (ANIPOL) or commanded direction polarity (CMDDIR). To ascertain the offset ANI input value, as affected by SCLD, PSET, ANIPOL or CMDDIR, use the TPANI command or the TFB command.

To determine the analog value from a specific input, precede the TANI command with the number of the input (e.g., 1TANI, 2TANI, etc.).

The TANI value is derived from the voltage applied to the corresponding analog input and ground. The analog value is determined from a 14-bit analog-to-digital converter (ADC). The minimum voltage response is -10.000VDC, the maximum voltage response is +10.000VDC.

TANV — Transfer Analog Input Voltage

Type	Transfer
Syntax	<!><i>TANV
Units	i = analog input number
Range	1 - 4 (AT6n00 & AT6n50) or 1 - 3 (620n, 625n & 6270)
Default	n/a
Response	TANV: *TANV1.963,1.453,0.444,0.112 1TANV: *1TANV1.963
See Also	[ANV], ANVO, ANVOEN, JOY, TINO

Product	Rev
AT6n00	1.0
OEM-AT6400	n/a
AT6n50	1.0
610n	n/a
615n	n/a
620n	1.0
625n	1.0
6270	1.0

The Transfer Analog Input Voltage (TANV) command returns the voltage level at the joystick analog inputs, referenced to ground. When using TANV, an analog input channel specifier can precede the TANV command. The analog channel specifier can be 1, 2, 3, or 4 (1TANV, 2TANV, 3TANV, or 4TANV). The response to the TANV command will be a voltage value returned from the analog channel queried. The value is derived from an 8-bit analog-to-digital converter with a range of 0-2.5VDC.

Pin # on the 25-pin Joystick Connector	Function
1	Analog Channel 1
2	Analog Channel 2
3	Analog Channel 3
4	Analog Channel 4 *(AT6n00 & AT6450)*
8	Shield
14	Ground

Pin # on 25-pin Joystick Connector	Function
15	Axes Select
16	Velocity Select
17	Joystick Release
18	Joystick Trigger
19	Joystick Auxiliary
23	+5VDC (out)

TAS — Transfer Axis Status

Type	Transfer
Syntax	<!><a>TAS<.i>
Units	i = bit location on the specified axis (See below)
Range	1 - 32
Default	n/a
Response	TAS: *TAS 0000_0000_0000_0000_0000_0000_0000_0000 * 0000_0000_0000_0000_0000_0000_0000_0000 * 0000_1000_0000_0000_0000_0000_0000_0000 * 0000_1000_0000_0000_0000_0000_0000_0000 1TAS: *1TAS0000_0000_0000_0000_0000_0000_0000_0000 (axis 1 status) TAS.5: *0011 (bit 5 of all four axes status registers) 1TAS.5: *0 (bit 5 of status register for axis 1)
See Also	[AS], [ASX], COMEXP, DRFLVL, ENC, EPM, ESTALL, GOWHEN, HOM, INFEN, JOG, JOY, LDTUPD, MA, MC, SMPER, STREAM, STRGTD, STRGTE, STRGTT, STRGTV, TASF, TASX, TSTAT

Product	Rev
AT6n00	1.4
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.5
625n	1.0
6270	1.0

The Transfer Axis Status (TAS) command returns the current status of all axes. The response for TAS is as follows (**Note**: response is product dependent):

```
*TAS bbbb_bbbb_bbbb_bbbb_bbbb_bbbb_bbbb_bbbb  <- Axis 1
*    bbbb_bbbb_bbbb_bbbb_bbbb_bbbb_bbbb_bbbb  <- Axis 2
*    bbbb_bbbb_bbbb_bbbb_bbbb_bbbb_bbbb_bbbb  <- Axis 3
*    bbbb_bbbb_bbbb_bbbb_bbbb_bbbb_bbbb_bbbb  <- Axis 4
     ^                                     ^
   Bit #1                                Bit #32
```

FULL-TEXT STATUS REPORT AVAILABLE

The TAS status command reports a binary bit report. If you would like to see a more descriptive text-based report, use the TASF command description.

Bit # (left to right)	Function (1/Ø)	AT6n00	OEM-AT6n00	AT6250	AT6450	610n	615n	620n	625n	6270
1	Moving/Not Moving	x	x	x	x	x	x	x	x	x
2	Negative/positive-direction	x	x	x	x	x	x	x	x	x
3	Accelerating/Not Accelerating	x	x	x	x	x	x	x	x	x
4	At Velocity/Not at Velocity	x	x	x	x	x	x	x	x	x
5	Home Successful (HOM) YES/NO	x	x	x	x	x	x	x	x	x
6	Absolute/Incremental (MA)	x	x	x	x	x	x	x	x	x
7	Continuous/Preset (MC)	x	x	x	x	x	x	x	x	x
8	Jog Mode/Not Jog Mode (JOG)	x	x	x	x	n/a	n/a	x	x	x
9	Joystick Mode/Not Joystick Mode (JOY)	x	n/a	x	x	n/a	n/a	x	x	x
10	Encoder Step Mode/Motor Step Mode (ENC)	x	n/a	n/a	n/a	x	n/a	x	n/a	n/a
11	Position Maintenance (EPM) ON/OFF	x	n/a	n/a	n/a	x	n/a	x	n/a	n/a
12	Stall Detected (ESTALL) YES/NO	x	n/a	n/a	n/a	x	n/a	x	n/a	n/a
13	Drive Shut Down YES/NO	x	n/a	x	x	x	x	x	x	x
14 *	Drive Fault occurred YES/NO	x	n/a	x	x	x	x	x	x	x
15	Positive-direction Hardware Limit Hit YES/NO	x	x	x	x	x	x	x	x	x
16	Negative-direction Hardware Limit Hit YES/NO	x	x	x	x	x	x	x	x	x

Bit # (left to right)	Function (1/Ø)	AT6n00	OEM-AT6n00	AT6250	AT6450	610n	615n	620n	625n	6270
17	Positive-direction Software Limit Hit YES/NO	x	x	x	x	x	x	x	x	x
18	Negative-direction Software Limit Hit YES/NO	x	x	x	x	x	x	x	x	x
19	Within Deadband (EPMDB) YES/NO	x	n/a	n/a	n/a	x	n/a	x	n/a	n/a
20	In Position (COMEXP) YES/NO	x	n/a	n/a	n/a	n/a	n/a	x	n/a	n/a
21	Distance Streaming Mode (STREAM1) YES/NO	x	x	n/a	n/a	n/a	n/a	n/a	n/a	n/a
22	Velocity Streaming Mode (STREAM2) YES/NO	x	x	n/a	n/a	n/a	n/a	n/a	n/a	n/a
23	Position Error Exceeded (SMPER) YES/NO	n/a	n/a	x	x	n/a	x	n/a	x	x
24 **	In Target Zone (STRGTD & STRGTV) YES/NO	x	n/a	x	x	x	x	x	x	x
25	Target Zone Timeout occurred (STRGTT) YES/NO	x	n/a	x	x	x	x	x	x	x
26 ***	Motion suspended pending GOWHEN YES/NO	x	x	x	x	x	x	x	x	x
27	LDT Position Read Error YES/NO	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	x
28 ****	Registration move initiated by trigger since last GO command	x	x	x	x	x	x	x	x	x
29	RESERVED	---	---	---	---	---	---	---	---	---
30	Pre-emptive (OTF) GO or Registration profile not possible	x	x	x	x	x	x	x	x	x
31	RESERVED	---	---	---	---	---	---	---	---	---
32	RESERVED	---	---	---	---	---	---	---	---	---

* The input functions must be enabled (INFEN1) before a drive fault will be recognized.
610n only: TASX bits 1-3 provide specific causes for the fault.
** This bit is set only after the successful completion of a move.
*** This bit is cleared if GOWHEN condition is true, or if STOP (!S) or KILL (!K or ^K) is executed.
**** This bit is cleared with the next GO command.

TASF — Transfer Axis Status (full-text report)

Type	Transfer
Syntax	<!><a>TASF
Units	n/a
Range	n/a
Default	n/a
Response	TASF: (see example below)
See Also	[AS], [ASX], COMEXP, DRFLVL, ENC, EPM, ESTALL, GOWHEN, HOM, INFEN, JOG, JOY, LDTUPD, MA, MC, SMPER, STREAM, STRGTD, STRGTE, STRGTT, STRGTV, TAS, TASX, TSTAT

Product	Rev
AT6n00	1.4
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.5
625n	1.0
6270	1.0

The TASF command returns a text-based status report of all axes. This is an alternative to the binary report (TAS).

Example TASF response (for 610n):

```
*TASF              AXIS #                      AXIS #
*                    1                           1
*Moving              NO      POS Sftwr Limit Hit NO
*Direction NEG       NO      NEG Sftwr Limit Hit NO
*Accelerating        NO      Within Deadband     YES
*At Velocity         NO      In Position         NO
*
*Home successful     NO      Distance Streaming  NO
*Mode Absolute       NO      Velocity Streaming  NO
*Mode Continuous     NO      Pos Error Exceeded  NO
*Jog Mode            NO      In Target Zone      YES
*
*Joystick Mode       NO      Target Zone Timeout NO
*Encoder Step Mode   NO      Gowhen is Pending   NO
*Position Maint Mode NO      LDT Failed/Bad Read NO
*Stall Detected      NO      Reg Move Commanded  NO
*
*Drive Shutdown      NO      RESERVED
*Drive Faulted       NO      Preset Move OvershotNO
*POS Hard Limit Hit  NO
*NEG Hard Limit Hit  NO
```

TASX — Transfer Extended Axis Status

Type	Transfer
Syntax	<!><a>TASX<.i>
Units	i = bit location on the specified axis (See below)
Range	1 - 32
Default	n/a
Response	TASX: *TASX 0000_0000_0000_0000_0000_0000_0000_0000 1TASX: *1TASX0000_0000_0000_0000_0000_0000_0000_0000 (axis 1 status) 1TASX.3: *0 (bit 3 of status register for axis 1)
See Also	[AS], [ASX], [ER], TAS, TASXF, TER

Product	Rev
AT6n00	4.1
AT6n50	4.1
610n	4.0
615n	4.1
620n	4.1
625n	4.1
6270	4.1

The Transfer Extended Axis Status (TASX) command returns the current status for axis 1 of the 610n. The response for TASX is as follows (**Note**: response is product dependent):

```
*TASX bbbb_bbbb_bbbb_bbbb_bbbb_bbbb_bbbb_bbbb  <- Axis 1
      ^                                    ^
   Bit #1                               Bit #32
```

FULL-TEXT STATUS REPORT AVAILABLE

The TASX status command reports a binary bit report. If you would like to see a more descriptive text-based report, use the TASXF command description.

Bit Assignment (left to right)	Function (1 = yes, Ø = no)	AT6n00	OEM-AT6n00	AT6n50	610n	615n	620n	625n	6270
1	Motor Fault (610n only)	---	---	---	x	---	---	---	---
2	Low Voltage (610n only)	---	---	---	x	---	---	---	---
3	Over Temperature Fault (610n only)	---	---	---	x	---	---	---	---
4	Drive Fault Input Active *	x	x	x	x	x	x	x	x
5-32	RESERVED	---	---	---	---	---	---	---	---

* Bit #4 indicates the current hardware state of the drive fault input, whether or not the drive is enabled.

TASXF — Transfer Extended Axis Status, (full-text report)

Type	Transfer
Syntax	<!><a>TASXF
Units	n/a
Range	n/a
Default	n/a
Response	TASXF: (see example below)
See Also	[AS], [ASX], [ER], TAS, TASX, TER

Product	Rev
AT6n00	4.1
AT6n50	4.1
610n	4.0
615n	4.1
620n	4.1
625n	4.1
6270	4.1

The TASXF command returns a text-based status report of all axes. This is an alternative to the binary report (TASX).

Example response (for 610n):

```
*TASF              AXIS #
*                    1
*Motor Fault         NO
*low-voltage         NO
*Over-temperature    NO
*Drive Fault Active  NO
```

TCA — Transfer Value of Captured ANI Input

Type	Transfer
Syntax	<!><a>TCAc
Units	c = letter of trigger input
Range	Range of the response is -10V to +10V
Default	n/a
Response	TCAA: *TCAA+0,+0,+0,+0 1TCAA: *1TCAA+0
See Also	[ANI], [CA], INFNC, [PCA], SFB, [SS], SSFR, TANI, TPCA, TSS

Product	Rev
AT6n00	n/a
AT6n50-ANI	3.3
610n	n/a
615n-ANI	3.3
620n	n/a
625-ANI	3.3
6270-ANI	3.0

The TCA command displays the current captured ANI value (volts). After displaying the captured ANI value, the respective capture status bit (reported with the TSS or SS commands) is cleared, but the voltage information remains in the register until it is overwritten by a subsequent capture from the trigger input.

The ANI value is referenced in volts and does not reflect the affects of scaling (SCLD), position offset (PSET), ANI feedback polarity (ANIPOL), or commanded direction polarity (CMDDIR). To assign/compare the ANI input value as affected by SCLD, PSET, ANIPOL, or CMDDIR, use the TPCA command.

The ANI input value can be captured only by a trigger input signal (trigger A, B, C or D). The appropriate trigger inputs must be defined as *trigger interrupt* inputs with the INFNCi-H command, where i can be 25, 26, 27 or 28, representing trigger inputs A, B, C or D, respectively. Once defined, an active signal on the specified trigger input will capture the ANI values on all axes. The ANI *voltage* information is stored in registers and is available at the next 1-ms update through the use of the CA and TCA commands.

CAPTURE ACCURACY

If ANI feedback is selected with the SFB command, the captured ANI value is interpolated from the last sampled ANI input value and rate of change of the ANI input value, and the time elapsed since the last sample. The sample rate is determined by the SSFR and INDAX commands (*system update rate*). The accuracy of the capture is ±50µs x velocity.

If ANI feedback is NOT selected with the SFB command, the last sampled ANI value is simply stored as the captured ANI value. The accuracy is one system update period (determined by SSFR and INDAX).

Response for TCAA: *TCAAr,r,r,r where r is ANI volts
Response for 1TCAA: *1TCAAr where r is ANI volts

TCMDER Transfer Command Error

Type	Transfer or Program Debug Tool
Syntax	<!>TCMDER
Units	n/a
Range	n/a
Default	n/a
Response	TCMDER: *(incorrect command)
See Also	ERRBAD, [SS], TSS

Product	Rev
AT6n00	2.1
AT6n50	1.0
610n	4.0
615n	1.0
620n	2.1
625n	1.1
6270	1.0

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

When the bad command is detected, the controller sends an error message to the screen, followed by the ERRBAD error prompt (?). To determine which command is in error, enter the TCMDER command and the controller will display the command, including all its command fields, if any.

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

Example:

```
DEF badprg        ; Begin definition of program called badprg
MA11              ; Select the absolute preset positioning mode
A25,40            ; Set acceleration
AD11,26           ; Set deceleration
V5,8              ; Set velocity
VAR1=0            ; Set variable #1 equal to zero
GO11              ; Initiate move on both axes
IF(VAR1<)16       ; Mistyped IF statement—should be typed as: IF(VAR1<16)
VAR1=VAR1+1       ; If variable #1 is less than16, increment the counter by 1
NIF               ; End IF statement
END               ; End programming of program called badprg
RUN badprg        ; Run the program called badprg
                  ; (this will cause an error --see comment box below)
; ********************************************************************
; * 1. When you run the badprg program, you should see this error      *
; *    message on your screen: "*INCORRECT DATA"  (this error message  *
; *    indicates incorrect command syntax).                            *
; * 2. Type "TCMDER" and press enter.  This queries the controller to  *
; *    display the command that caused the error.  In this case, the   *
; *    response will be "*IF(VAR1<)16".                                *
; ********************************************************************
```

TCNT Transfer Hardware Counter Value

Type	Transfer
Syntax	<!><a>TCNT
Units	n/a
Range	n/a
Default	n/a
Response	TCNT: *TCNT+0,+0,+0,+0 1TCNT: *1TCNT+0
See Also	[CNT], CNTE, CNTINT, CNTR

Product	Rev
AT6n00	1.0
OEM-AT6n00	n/a
AT6n50	n/a
610n	4.0
615n	n/a
620n	1.0
625n	n/a
6270	n/a

The Transfer Hardware Counter Value (TCNT) command returns the current value of the hardware counter.

The hardware counter is one of the encoder ports converted to a hardware counter through the use of the CNTE command. The hardware counter will count up or down. The direction of count is specified by the signal on the encoder channel B+ and B- connections. A positive differential signal, when measured between B+ and B-, will infer a negative count direction. A negative differential signal, when measured between B+ and B-, will infer a positive count direction. The count itself is determined from the signal on

A+ and A-. Each count is registered on the positive (rising) edge of a transition for a signal measured between A+ and A-. To reset the counter, apply a signal to Z+ and Z-, or issue the command CNTR.

For all encoder channels not defined as counters, the TCNT command will report a count value of zero.

TDAC — Transfer Digital-to-Analog Converter (DAC) Voltage

Type	Transfer
Syntax	<!><@><a>TDAC
Units	Reported value represents volts
Range	Range of reported value is -10 to +10
Default	n/a
Response	TDAC: *TDAC10.000,10.000 1TDAC: *1TDAC10.000
See Also	[DAC], DACLIM, DACMIN, SFB, SGAF, SGI, SGP, SGV, SGVF, SOFFS, SOFFSN

Product	Rev
AT6n00	n/a
AT6n50	1.0
610n	n/a
615n	1.0
620n	n/a
625n	1.0
6270	1.0

This command allows you to display the voltage being commanded at the digital-to-analog converter (DAC). This is the *analog command signal* (plus any voltage offset set with the SOFFS and SOFFSN commands) output by the servo controller. The DAC output is a 12-bit, ± 10V analog signal. At any point, the voltage that is currently being commanded can be displayed using the TDAC command. If direct control over the analog voltage is required, it can be accomplished by setting the servo algorithm gains (SGP, SGI, SGV, SGVF, & SGAF) to zero and using the SOFFS and SOFFSN commands.

When checking the DAC value (counts) in the Fast Status report (see FASTAT), note that there are 2048 counts/volt.

Example:

```
TDAC        ; Display the actual output voltage for each axis.
            ; Example response is: *TDAC4.552,5.552,5.552,5.552
```

TDIR — Transfer Program Directory

Type	Transfer
Syntax	<!>TDIR
Units	n/a
Range	n/a
Default	n/a
Response	TDIR: *NO PROGRAMS DEFINED *33000 OF 33000 BYTES (100%) PROGRAM MEMORY REMAINING *500 of 500 SEGMENTS (100%) COMPILED MEMORY REMAINING
See Also	DEF, INFNC, MEMORY, [SEG] TMEM, TSEG

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Transfer Program Directory (TDIR) command returns the names of all the programs and subroutines defined with the DEF command, and the amount of memory each consumes. The format of the response is as follows:

```
*1 - PROG1 USES 345 BYTES
*2 - PROG2 USES 333 BYTES
*32322 OF 33000 BYTES (98%) PROGRAM MEMORY REMAINING
*500 of 500 SEGMENTS (100%) COMPILED MEMORY REMAINING
```

(In the above example, PROG1 and PROG2 are names of programs.)

NOTE: The amount of memory available is product-dependent.

The number in front of the program name is the number to use when defining specific inputs (INFNC) to correspond to a specific program (function P of INFNC), or when programs are selected via BCD (function B of INFNC).

If the program is intended to be a compiled profile and has been successfully compiled (PCOMP), then the line item for the program is amended with "(COMPILED AS A PATH)".

TDPTR — Transfer Data Pointer Status

Type	Data Storage
Syntax	<!>TDPTR
Units	n/a
Range	n/a
Default	n/a
Response	TDPTR *TDPTR1,1,1
See Also	DATPTR, DATSIZ, [DPTR]

Product	Rev
AT6n00	2.2
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The TDPTR command responds with a 3-integer status report (i,i,i). The first integer is the number of the current active data program (the program number specified with the last DATSIZ or DATPTR command). The second integer is the location number of the data element to which the data pointer is currently pointing. The third integer is the increment set with the last DATPTR command.

The DPTR command can be used to compare the current pointer location against another value or variable, or to assign the pointer location number to a variable.

Example

```
DATSIZ4,200     ; Create data program called DATP4 with 200 data elements
DATPTR4,20,2    ; Set the data pointer to data element #20 in DATP4 and set the
                ; increment to 2 (DATP4 becomes the current active data program)
TDPTR           ; Response is *TDPTR4,20,2. Indicates that the data pointer is
                ; pointing to data element #20 in data program #4 (DATP4),
                ; and the increment setting is 2.
```

TER — Transfer Error Status

Type	Transfer
Syntax	<!>TER<.i>
Units	i = specific error status bit
Range	1 - 32
Default	n/a
Response	TER: *TER0000_0000_0000_0000_0000_0000_0000_0000 TER.4: *0 (bit 4 of error status register)
See Also	[ASX], DRFLVL, [ER], ERROR, ESTALL, GOWHEN, INFEN, INFNC, LDTUPD, LH, LS, SMPER, STRGTT, TASX, TCMDER, TERF

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Transfer Error Status (TER) command returns the status of the 32 error bits. There is only one error status for all axes.

NOTE

The specific error bits must be enabled by the Error Enable (ERROR) command before the TER command will provide the correct status of the error conditions.

TER response: *TERbbbb_bbbb_bbbb_bbbb_bbbb_bbbb_bbbb_bbbb

Bit #1 ... Bit #32

FULL-TEXT STATUS REPORT AVAILABLE

The TER status command reports a binary bit report. If you would like to see a more descriptive text-based report, use the TERF command description.

The function of each axis status bit is shown below.

Bit #	**Function** (1 = Yes; Ø = No)
1*	Stall Detected: Functions when stall detection has been enabled (ESTALL). (n/a for OEM-AT6n00)
2	Hard Limit Hit: Functions when hard limits are enabled (LH).
3	Soft Limit Hit: Functions when soft limits are enabled (LS).
4	Drive Fault: Detected only of the drive is enabled (DRIVE) and the drive fault level is set correctly (DRFLVL and INFEN). (Drive Fault monitoring is n/a for OEM-AT6n00.) 610n: use ASX, TASX or TASXF to determine exact cause (ASX status is checked even if the drive is disabled).
5	RESERVED (refer to the ERROR command)
6	Kill Input: When an input is defined as a Kill input (INFNCi-C), and that input becomes active.
7	User Fault Input: When an input is defined as a User Fault input (INFNCi-F), and that input becomes active.
8	Stop Input: When an input is defined as a Stop input (INFNCi-D), and that input becomes active.
9	Stepper products—Pulse Cutoff (P-CUT): When the pulse cutoff input is activated (not grounded). Servo products—Enable input (ENBL): When the enable input is activated (not grounded). (n/a for OEM-AT6n00)
10	Pre-emptive (on-the-fly) GO or registration move profile not possible.
11**	Target Zone Settling Timeout Period (set with the STRGTT command) is exceeded.
12**	Maximum Position Error (set with the SMPER command) is exceeded.
13	RESERVED
14	Position relationship in GOWHEN already true when GO, GOL, FSHFC, or FSHFD was executed.
15***	LDT Position Read Error: Can be caused by LDT not connected, mechanical failure of LDT or LDTUPD command value too low.
16-32	RESERVED

* Stepper products only
** Servo products only
*** 6270 only

When error bit 5 (Commanded Kill or Stop) of the ERROR command is enabled (ERROR.5-1), a Stop (!S) or a Kill (!K or <ctrl>K) command will cause the controller to GOSUB or GOTO to the error program (ERRORP). Within the error program the cause of the error will need to be determined. The transfer error status (TER) command can be used to determine the cause of the error. If none of the error status bits are set, the cause of the error is a commanded kill or a commanded stop. The reason for not setting a bit on this error condition is that there is no way to clear the error condition upon leaving the error program.

TERF — Transfer Error Status (full-text report)

		Product	Rev
Type	Transfer		
Syntax	<!>TERF	AT6n00	1.0
Units	n/a	AT6n50	1.0
Range	n/a	610n	4.0
Default	n/a	615n	1.0
Response	TERF: (see example below)	620n	1.0
See Also	[ASX], DRFLVL, [ER], ERROR, ESTALL, GOWHEN, INFEN, INFNC, LDTUPD, LH, LS, SMPER, STRGTT, TASX, TCMDER, TER	625n	1.0
		6270	1.0

The TERF command returns a text-based status report of all axes. This is an alternative to the binary report (TER).

Example TERF response (for 610n):

```
*TERF
*Stall Detected      NO
*Hard Limit Hit      NO
*Soft Limit Hit      NO
*Drive Fault Active  NO
*
*Reserved            NO
*Kill Input Active   NO
*User Fault Input    NO
*Stop Input Active   NO
*
*Pulse Cutoff OK     NO
*Profile impossible  NO
*Target Zone Timeout NO
*Max Position Error  NO
*
*GOWHEN cond true    NO
```

TEST — Test Motion

Type	Motion
Syntax	<!>TEST
Units	n/a
Range	n/a
Default	n/a
Response	n/a
See Also	A, AD, D, GO, K, MA, MC, PSET, S, SSV, V

Product	Rev
AT6n00	1.0
AT6n50	n/a
610n	4.0
615n	n/a
620n	1.0
625n	n/a
6270	n/a

The Test Motion (TEST) command initiates a 25000-step move positive-direction on axis 1, followed by a 1-second time delay, followed by a 25000-step move negative-direction on axis 1. This motion is repeated for all axes. The velocity is set to 25000 steps/sec and the acceleration and deceleration are set to 250000 steps/sec/sec.

WARNING

This command overrides the end-of-travel limits (LH and LS) settings, therefore, this command should only be used during setup, while the motor is uncoupled from the load.

NOTE

Do not use the TEST command from the Panel Mode in Motion Architect.

TEX — Transfer Program Execution Status

Type	Transfer
Syntax	!TEX
Units	n/a
Range	n/a
Default	n/a
Response	!TEX: *PROGRAM NOT EXECUTING
See Also	DEF

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Transfer Program Execution Status (TEX) command reports the status of any programs in progress.

If the program PAUL was in progress, and within that program a loop was in progress, the response to !TEX could look like the following: *PROGRAM=PAUL COMMAND=LN LOOP COUNT=12

TFB — Transfer Position of Selected Feedback Devices

Type	Transfer
Syntax	<!><@><a>TFB
Units	response is position of the selected feedback devices
Range	n/a
Default	n/a
Response	TFB *TFB+0,+0,+0,+0 1TFB *1TFB+0
See Also	[ANI], ANIPOL, CMDDIR, ENCPOL, [FB], [LDT], LDTPOL, [PANI], [PE], PSET, SCALE, SCLD, SFB, TANI, TLDT, TPANI, TPCE, TPCL, TPE

Product	Rev
AT6n00	n/a
AT6n50	1.0
610n	n/a
615n	1.0
620n	n/a
625n	3.0
6270	1.0

Use the TFB command to return the current values of the feedback sources selected with the SFB command. If you do not change the default SFB selection, the response will indicate LDT position for the 6270, and encoder position for the AT6n50 and 625n, and internal resolver position for 615n.

If scaling is **not** enabled, the position values returned will be counts. If scaling is enabled (SCALE1), the values will be scaled by the SCLD value.

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

Example:

```
SFB2,1      ; Select ANI feedback on axis 1 and encoder feedback on axis 2
TFB         ; Report ANI input #1's voltage and encoder #2's position.
            ; Sample response is *TFB4.256,2.436
```

TFS — Transfer Following Status

Type	Following and Transfer
Syntax	<!><a>TFS
Units	n/a
Range	n/a
Default	n/a
Response	TFS *TFS0000_0000_0000_0000_00 * 0000_0000_0000_0000_00 * 0000_0000_0000_0000_00 * 0000_0000_0000_0000_00 1TFS *1TFS0000_0000_0000_0000_00
See Also	FMCLEN, FMCP, FOLEN, FOLMAS, FPPEN, [FS], FSHFC, FSHFD, MC, [NMCY], [PMAS], TFSF

Product	Rev
AT6n00	3.0
AT6n50	3.0
610n	4.0
615n	3.0
620n	3.0
625n	3.0
6270	3.0

The Transfer Following Status (TFS) command returns the current Following status of all axes. The response for TFS is as follows (**Note:** response is product dependent):

```
*TFS bbbb_bbbb_bbbb_bbbb_bb <- Axis 1
*    bbbb_bbbb_bbbb_bbbb_bb <- Axis 2
*    bbbb_bbbb_bbbb_bbbb_bb <- Axis 3
*    bbbb_bbbb_bbbb_bbbb_bb <- Axis 4
     ^                   ^
  Bit #1              Bit #18
```

FULL-TEXT STATUS REPORT AVAILABLE

The TFS status command reports a binary bit report. If you would like to see a more descriptive text-based report, use the TFSF command description.

Bit Assignment (left to right)	**Function** (YES = 1; NO = Ø)	
1	Slave in Ratio Move	A Following move is in progress.
2	Ratio is Negative	The current ratio is negative (i.e., the slave counts are counting in the opposite direction from the master counts).
3	Slave Ratio Changing	The slave is ramping from one ratio to another (including a ramp to or from zero ratio).
4	Slave At Ratio	The slave is at constant non-zero ratio.
	Bits 1-4 indicate the status of slave Following motion. They mimic the meaning and organization of Axis Status (TAS & AS) bits 1-4, except that each bit indicates the current state of the ratio, rather than the current state of the velocity.	
* 5	FOLMAS Active	A master is specified with the FOLMAS command.
* 6	FOLEN Active	Following has been enabled with the FOLEN command.
* 7	Master is Moving	The specified master is currently in motion.
8	Master Dir Neg	The current master direction is negative. (Bit must be cleared to allow Following move in preset mode–MCØ).
	Bits 5-8 indicate the status required for Following motion (i.e., a master must be assigned, Following must be enabled, the master must be moving, and for many features, the master direction must be positive). Unless the master is a commanded position of another axis, it is likely that minor vibration of the master will cause bits 7-8 to toggle on and off, even if the master is nominally "at rest". These bits are meant primarily as a quick diagnosis for the absence of master motion, or master motion in the wrong direction. Many features require positive master counting to work properly.	
9	OK to Shift	Conditions are valid to issue shift commands (FSHFD or FSHFC).
10	Shifting now	A shift move is in progress.
11	Shift is Continuous	An FSHFC-based shift move is in progress.
12	Shift Dir is Neg	The direction of the shift move in progress is negative.
	Bits 9-12 indicate the shift status of the slave. Shifting is super-imposed motion, but if viewed alone, can have its own status. In other words, bits 10-12 describe only the shifting portion of motion.	
13	Master Cyc Trig Pend	A master cycle restart is pending the occurrence of the specified trigger.
14	Mas Cyc Len Given	A non-zero master cycle length has been specified with the FMCLEN command.
15	Master Cyc Pos Neg	The current master cycle position (PMAS) is negative. This could be by caused by a negative initial master cycle position (FMCP), or if the master is moving in the negative direction.
16	Master Cyc Num > 0	The master position (PMAS) has exceeded the master cycle length (FMCLEN) at least once, causing the master cycle number (NMCY) to increment.
	Bits 13-16 indicate the status of master cycle counting. If a Following application is taking advantage of master cycle counting, these bits provide a quick summary of some important master cycle information.	
17	Mas Pos Prediction On	Master position prediction has been enabled (FPPEN).
18	Mas Filtering On	A non-zero value for master position filtering (FFILT) is in effect.
	Bit 17-18 indicate the status of master position measurement features.	

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

TFSF — Transfer Following Status (full-text report)

Type	Following and Transfer
Syntax	<!><a>TFSF
Units	n/a
Range	n/a
Default	n/a
Response	TFSF: (see example below)
See Also	FMCLEN, FMCP, FOLEN, FOLMAS, FPPEN, [FS], FSHFC, FSHFD, MC, [NMCY], [PMAS], TFS

Product	Rev
AT6n00	3.0
AT6n50	3.0
610n	4.0
615n	3.0
620n	3.0
625n	3.0
6270	3.0

The TFSF command returns a text-based status report of all axes. This is an alternative to the binary report (TFS).

Example response (for 610n):

```
*TFSF              AXIS #                    AXIS #
*                     1                         1
*Slave in Ratio Move NO      Pos Prediction On   YES
*Ratio is Negative   NO      Master Filtering On NO
*Slv Ratio Changing  NO
*Slave At Ratio      NO
*
*Folmas Active       NO
*Folen Active        NO
*Master is Moving    NO
*Master Dir Neg      NO
*
*OK to Shift         NO
*Shifting now        NO
*Shift is Continuous NO
*Shift Dir is Neg    NO
*
*Master Cyc Trig Arm NO
*Mas Cyc Len Given   NO
*Master Cyc Pos Neg  NO
*Master Cyc Num > 0  NO
```

TGAIN — Transfer Servo Gains

Type	Transfer
Syntax	<@><a>TGAIN
Units	n/a
Range	n/a
Default	n/a
Response	TGAIN: *SGP1,2,3,4 *SGI.1,.1,0,0 *SGV25,25,40,40 *SGVF100,100,100,100 *SGAF0,0,0,0 1TGAIN: *1SGP1 *1SGI.1 *1SGV25 *1SGVF100 *1SGAF0
See Also	SFB, SGAF, SGAFN, SGENB, SGI, SGILIM, SGIN, SGP, SGPN, SGSET, SGV, SGVF, SGVFN, SGVN, SOFFS, SOFFSN, SSWG, TSGSET

Product	Rev
AT6n00	n/a
AT6n50	1.0
610n	n/a
615n	1.0
620n	n/a
625n	1.0
6270	3.0

This command allows you to display the current value of each of the control algorithm gains (SGP, SGI, SGV, SGAF, & SGVF). Each time an individual gain is entered, the current value is updated to be that value. When a gain set is enabled with the SGENB command, the current value of each gain is set to the values saved in that particular gain set.

NOTE

Tuning gains are specific to the feedback source that was in use (selected with the last SFB command) at the time the gains were established with the respective gain commands (SGI, SGP, etc.).

Example:

```
SGP5,5,10,10    ; Set the gains for the proportional gain
SGI.1,.1,0,0    ; Set the gains for the integral gain
SGV50,60,0,0    ; Set the gains for the velocity gain
SGVF5,6,10,11   ; Set the gains for the velocity feedforward gain
SGAF0,0,0,0     ; Set the gains for the acceleration feedforward gain
TGAIN           ; Display current values for all gains.  Example response:
                ;  *SGP5,5,10,10
                ;  *SGI.1,.1,0,0
                ;  *SGV50,60,0,0
                ;  *SGVF5,6,10,11
                ;  *SGAF0,0,0,0
```

[TIM] Current Timer Value

		Product	Rev
Type	Assignment or Comparison	AT6n00	1.0
Syntax	See below	AT6n50	1.0
Units	Milliseconds	610n	4.0
Range	Maximum count is 999,999,999 (approx. 11 days, 13 hours)	615n	1.0
Default	n/a	620n	1.0
Response	n/a	625n	1.0
See Also	TIMINT, TIMST, TIMSTP, TTIM	6270	1.0

The Current Timer Value (TIM) command is used to assign the timer value to a variable, or to make a comparison against another value. The value returned is in milliseconds.

Syntax: VARx=TIM where x is a numeric variable number, or TIM can be used in an expression such as IF(TIM<2400)

Example:
```
VAR1=TIM        ; Timer value is assigned to variable 1
IF(TIM<1000)    ; If timer value is < 1000 milliseconds, do the IF statement
VAR1=TIM + 10   ; Timer value plus 10 assigned to variable 1
NIF             ; End IF statement
```

TIMINT Timer Value to Interrupt PC-AT

		Product	Rev
Type	Timer	AT6n00	1.0
Syntax	<!>TIMINT<b>,<i>	AT6n50	1.0
Units	i = milliseconds	610n	n/a
Range	b = 0 (reset and start) or 1 (stop), i = 0 - 999,999,999	615n	n/a
Default	0,0	620n	n/a
Response	TIMINT: *TIMINT0,0	625n	n/a
See Also	INTHW, [TIM], TIMST, TIMSTP, TTIM	6270	n/a

The Timer Value to Interrupt PC-AT (TIMINT) command sets the timer value upon which the 6000 controller will interrupt the PC-AT. The time value at which the interrupt will occur is specified by the second field in the command.

The TIMINT command also determines if the timer is to be stopped when the value is reached, or if the timer is to be reset and started again. If the timer is to be stopped upon reaching the interrupt value, a one should be specified for the first field. If the timer is to be reset and restarted upon reaching the interrupt value, a zero should be specified for the first field. By specifying a zero in the first field, an interrupt will occur repeatedly.

NOTE: Before an interrupt will occur, timer interrupt bit #25 must be enabled with the INTHW command.

Example:
```
INTHW.25-1      ; Set timer interrupt bit
TIMINT1,10000   ; Interrupt PC-AT once after 10000 ms, do not restart the timer
TIMST0          ; Reset and start timer
```

TIMST Start Timer

		Product	Rev
Type	Timer	AT6n00	1.0
Syntax	<!>TIMST<b>	AT6n50	1.0
Units	n/a	610n	4.0
Range	b = 0 (reset and start) or 1 (start)	615n	1.0
Default	0	620n	1.0
Response	TIMST: No response, acts as if TIMST1 command was issued	625n	1.0
See Also	SSFR, [TIM], TIMINT, TIMSTP, TTIM	6270	1.0

The Start Timer (TIMST) command is used to start the timer. If (TIMST0) is specified, the timer will be reset to zero and started. If (TIMST1) is specified, the timer will be started without reset. By specifying TIMST1, the timer can also be restarted after the Stop Timer (TIMSTP) command has been issued. This command, in conjunction with the stop timer (TIMSTP) command, provides a timer that can be used to time internal or external events.

NOTE

Use the following table to determine the resolution of the timer, and to determine the delay created by executing the TIMST and TIMSTP commands.

PRODUCT TYPE	TIMER RESOLUTION	DELAY FOR EXECUTING TIMST AND TIMSTP IN COMBINATION*
Steppers	2 ms	4 - 6 ms
Servos	1 system update period	(see table in SSFR command description)* Be sure to factor this value into your final time value.

If the timer is started and allowed to roll over the maximum timer count of 999,999,999 milliseconds (11 days, 13 hours, 46 minutes, 39.999 seconds), the timer will be stopped, and the value will be frozen at the maximum value.

Example:

```
TIMST0          ; Reset and start timer
GO1100          ; Initiate motion on axes 1 and 2
TIMSTP          ; Stop timer
TTIM            ; Transfer time required for move
```

TIMSTP Stop Timer

Type	Timer
Syntax	<!>TIMSTP
Units	n/a
Range	n/a
Default	n/a
Response	n/a
See Also	SSFR, [TIM], TIMINT, TIMST, TTIM

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Stop Timer (TIMSTP) command stops the timer. This command in conjunction with the start timer (TIMST) command, provides a timer that can be used to time internal or external events.

NOTE

Use the following table to determine the resolution of the timer, and to determine the delay created by executing the TIMST and TIMSTP commands.

PRODUCT TYPE	TIMER RESOLUTION	DELAY FOR EXECUTING TIMST AND TIMSTP IN COMBINATION*
Steppers	2 ms	4 - 6 ms
Servos	1 system update period	(see table in SSFR command description)* Be sure to factor this value into your final time value.

Example:

```
TIMST0          ; Reset and start timer
GO1100          ; Initiate motion on axes 1 and 2
TIMSTP          ; Stop timer
TTIM            ; Transfer time required for move
```

TIN — Transfer Input Status

		Product	Rev
Type	Transfer	AT6n00	1.0
Syntax	<!>TIN<.i>	AT6n50	1.0
Units	i = programmable input number	610n	4.0
Range	Product dependent	615n	1.0
Default	n/a	620n	1.0
Response	TIN: *TIN1111_0000_1111_0000_1111_0000_1111	625n	1.0
	TIN.4: *1 (status of input number 4)	6270	1.0
See Also	[IN], INFNC, INLVL, TINO, TLIM		

The Transfer Input Status (`TIN`) command returns the current status (active or inactive) of the programmable inputs. The input is *active* when it is grounded. The active level (active high or active low) for the inputs is established with the `INLVL` command. "High" means that current is flowing and no voltage is present at the input terminal; conversely, "low" means that no current is flowing and a voltage may be present at the input terminal. If the active level is set to active low (`INLVLØ` – default), the `TIN` response indicates active with a one (`1`) and inactive with a zero (`Ø`). If the active level is set to active high (`INLVL1`), the `TIN` response indicates active with a zero (`Ø`) and inactive with a one (`1`).

The general purpose programmable inputs are returned first, followed by the trigger inputs. Input bit assignments vary by product (see bit assignment table on page 6 of this document). The inputs are numbered 1 to *n* (*n* depends on the product) from left to right.

If a specific input is required, the bit number can be placed after the `TIN` command. For example, `TIN.5` would return bit 5, which corresponds to input 5.

TINO — Transfer Other Input Status

		Product	Rev
Type	Transfer	AT6n00	1.0
Syntax	<!>TINO<.i>	OEM-AT6n00	n/a
Units	i = number of input (see below)	AT6n50	1.0
Range	1 - 8	610n	4.0
Default	n/a	615n	4.0
Response	TINO: *TINO1111_0000	620n	1.0
	TINO.4: *1 (status of input number 4)	625n	1.0
See Also	[INO], JOY, TIN, TINOF	6270	1.0

The Transfer Other Input Status (`TINO`) command returns the status of all of the inputs not covered by the `TLIM` or `TIN` commands. These 8 additional inputs may be used for status feedback.

`TINO` response: `*TINObbbb_bbbb`

Bit #1 Bit #12

FULL-TEXT STATUS REPORT AVAILABLE

The `TINO` status command reports a binary bit report. If you would like to see a more descriptive text-based report, use the `TINOF` command description.

Bit	**Function** (1 = Yes, 0 = No)	Location
1	Joystick Auxiliary Input Active	Joystick Connector Pin 19
2	Joystick Trigger Input Active	Joystick Connector Pin 18
3	Joystick Axes Select Input Active	Joystick Connector Pin 15
4	Joystick Velocity Select Input High	Joystick Connector Pin 16
5	Joystick Release Input Active	Joystick Connector Pin 17
6	Pulse Cutoff Input — OK for motion Enable input — OK for motion	P-CUT terminal on AUX connector (steppers only) ENBL terminal on the AUX connector (servos only)
7	Not used, always 0	
8	Not used, always 0	

TINOF — Transfer Other Input Status (full-text report)

Type	Transfer
Syntax	<!>TINOF
Units	n/a
Range	n/a
Default	n/a
Response	TINOF: (see example below)
See Also	[INO], JOY, TIN, TINO

Product	Rev
AT6n00	1.0
OEM-AT6n00	n/a
AT6n50	1.0
610n	4.0
615n	4.0
620n	1.0
625n	1.0
6270	1.0

The TINOF command returns a text-based status report of all axes. This is an alternative to the binary report (TINO).

Example response (for 610n):

```
*TINOF
*Joy Aux In Active   NO
*Joy Trig In Active  NO
*Joy Axes Sel In Act NO
*Joy Vel Active High NO
*
*Joy Release In Act  NO
*Pulse Cutoff OK     YES
```

TINT — Transfer Interrupt Status

Type	Transfer
Syntax	<!>TINT<.i>
Units	i = hardware interrupt status bit number
Range	1 - 32
Default	n/a
Response	TINT: *TINT1111_0000_1111_0000_1111_0000_1111_0000 TINT.4: *1 (status of interrupt number 4)
See Also	INTCLR, INTHW, INTSW

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	n/a
615n	n/a
620n	n/a
625n	n/a
6270	n/a

The Transfer Interrupt Status (TINT) command returns the status of the hardware interrupt conditions. As soon as the interrupt status is read (TINT), the interrupts are cleared. If only one interrupt is to be transferred and cleared, use the bit select (.) and the corresponding bit number. For example, to transfer and clear interrupt bit number 13, type in TINT.13.

TINT response: *TINTbbbb_bbbb_bbbb_bbbb_bbbb_bbbb_bbbb_bbbb

Bit #1 ^ Bit #32 ^

Bit #	Function	Bit #	Function
1	Software Interrupt #1 (See INTSW)	17	Command Buffer Full
2	Software Interrupt #2	18 *	Pulse Cutoff (steppers) or Enable (servos) Activated
3	Software Interrupt #3	19	Program Complete
4	Software Interrupt #4	20 *	Drive Fault on any Axis
5	Software Interrupt #5	21	Reserved
6	Software Interrupt #6	22	Reserved
7	Software Interrupt #7	23	Limit Hit - hard or soft limit, on any axis
8	Software Interrupt #8	24 *	Stall Detected (steppers) or Position Error (servos) on any axis
9	Software Interrupt #9	25	Timer (TIMINT)
10	Software Interrupt #10	26 *	Counter (CNTINT) – steppers only
11	Software Interrupt #11	27	Input - any of the inputs defined by INFNCi-I
12	Software Interrupt #12	28	Command Error
13	Software Interrupt #13	29	Motion Complete on Axis 1
14	Software Interrupt #14	30	Motion Complete on Axis 2
15	Software Interrupt #15	31	Motion Complete on Axis 3 (AT6400 & AT6450)
16	Software Interrupt #16	32	Motion Complete on Axis 4 (AT6400 & AT6450)

* not applicable to the OEM-AT6n00

TLABEL Transfer Labels

		Product	Rev
Type	Transfer	AT6n00	1.0
Syntax	<!>TLABEL	AT6n50	1.0
Units	n/a	610n	4.0
Range	n/a	615n	1.0
Default	n/a	620n	1.0
Response	TLABEL: *NO LABELS DEFINED	625n	1.0
See Also	$	6270	1.0

The Transfer Labels (TLABEL) command returns the names of all the labels defined with the $ command.

The response to a TLABEL command if the labels call and open are defined in a program named prog1 is as follows:

```
*CALL DEFINED IN PROGRAM PROG1
*OPEN DEFINED IN PROGRAM PROG1
```

TLDT Transfer Position of LDT

		Product	Rev
Type	Transfer	AT6n00	n/a
Syntax	<!><a>TLDT	AT6n50	n/a
Units	Reported value is LDT counts or scaled (SCLD) units	610n	n/a
Range	n/a	615n	n/a
Default	n/a	620n	n/a
Response	TLDT *TLDT+0,+0 1TLDT *1TLDT+0	625n	n/a
See Also	[AS], CMDDIR, [ER], ERROR, [LDT], LDTPOL, PSET, SCALE, SCLD, SFB, TAS, TER, TFB	6270	1.0

Use the TLDT command to transfer the current LDT (linear displacement transducer) position.

If scaling is not enabled (SCALEØ), the value will represent actual LDT counts. If scaling is enabled (SCALE1), the value will be scaled by the distance scaling factor (SCLD).

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

An LDT position read error can be caused by a bad LDT connection, an LDT failure, or an LDTUPD command value being too small. If this error occurs, axis status bit #27 (reported with the TAS and AS commands) will be set. In addition, if ERROR bit #15 is enabled (ERROR.15-1), error status bit #15 (reported with the TER and ER commands) will also be set.

Example:

```
2TLDT           ; Report the position of the LDT for axis #2.
                ; Example response: *2TLDT+5.071
```

TLIM Transfer Limits

		Product	Rev
Type	Transfer	AT6n00	1.0
Syntax	<!><a>TLIM<.i>	AT6n50	1.0
Units	i = limit input number	610n	4.0
Range	Product dependent	615n	1.0
Default	n/a	620n	1.0
Response	TLIM: *TLIM110_011_001_100 TLIM.4: *0 (status of positive-direction limit input on axis 2)	625n	1.0
See Also	HOM, HOMLVL, LHLVL, [LIM], TAS, TIN, TINO, TOUT	6270	1.0

The Transfer Limits (TLIM) command returns the current hardware state of the limit inputs on all axes. There are 3 limit inputs per axis. To determine if an end-of-travel limit has been hit, refer to the TAS command response, bits 15 through 18.

TLIM response: *TLIMbbb_bbb_bbb_bbb

Bit #1 ^ Bit #12 ^

Descriptions of each bit function are defined in the table below:

Bit	Function
1	Axis 1 - Positive-Direction Limit
2	Axis 1 - Negative-Direction Limit
3	Axis 1 - Home Limit
4	Axis 2 - Positive-Direction Limit
5	Axis 2 - Negative-Direction Limit
6	Axis 2 - Home Limit
7	Axis 3 - Positive-Direction Limit (AT6400 and AT6450)
8	Axis 3 - Negative-Direction Limit (AT6400 and AT6450)
9	Axis 3 - Home Limit (AT6400 and AT6450)
10	Axis 4 - Positive-Direction Limit (AT6400 and AT6450)
11	Axis 4 - Negative-Direction Limit (AT6400 and AT6450)
12	Axis 4 - Home Limit (AT6400 and AT6450)

TMEM — Transfer Memory Usage

Type	Transfer
Syntax	<!>TMEM
Units	n/a
Range	n/a
Default	n/a
Response	TMEM: *33000 OF 33000 BYTES (100%) PROGRAM MEMORY REMAINING *500 OF 500 SEGMENTS (100%) COMPILED MEMORY REMAINING
See Also	DEF, MEMORY, PCOMP, [SEG], TDIR, TSEG

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Transfer Memory Usage (TMEM) command returns the amount of available memory for user program storage and for storing contouring path segments. A path segment is one element of the path (e.g., PLIN3777,3777). The amount of memory available can be modified with the MEMORY command. As programs are defined (DEF) and paths are compiled (PCOMP), the amount of memory available decreases.

> **NOTE**
> The amount of memory available for user program storage varies by product.

TNMCY — Transfer Master Cycle Number

Type	Following and Transfer
Syntax	<!><a>TNMCY
Units	n/a
Range	n/a
Default	n/a
Response	TNMCY *TNMCY0,0,0,0 1TNMCY *1TNMCY0
See Also	FMCLEN, FMCNEW, [FS], TRGFN, TFS

Product	Rev
AT6n00	3.0
AT6n50	3.0
610n	4.0
615n	3.0
620n	3.0
625n	3.0
6270	3.0

The Transfer Master Cycle Number (TNMCY) command displays the current master cycle number for all axes, or the axis specified. The value represents the current cycle number, not the position of the master (or the slave). The master cycle number is set to zero when master cycle counting is restarted, and is incremented each time a master cycle finishes (i.e., rollover occurs). It will often correspond to the number of complete parts in a production run. This value may be used for subsequent decision making, or simply recording the cycle number corresponding to some other event.

The master must be assigned first (FOLMAS command) before this command will be useful.

For a complete discussion of master cycles, please refer to the Following chapter in the *6000 Series Programmer's Guide.*

TOUT — Transfer Output Status

		Product	Rev
Type	Transfer	AT6n00	1.0
Syntax	<!>TOUT<.i>	AT6n50	1.0
Units	i = number of a specific programmable output	610n	4.0
Range	Product dependent	615n	1.0
Default	n/a	620n	1.0
Response	TOUT: *TOUT1111_0000_1111_0000_1111_0000 TOUT.4: *1 (status of output #4)	625n	1.0
See Also	OUT, OUTFNC, OUTLVL, TIN, TINO	6270	1.0

The Transfer Output Status (TOUT) command returns the current status (active or inactive) of the programmable outputs. The output is *active* when it is grounded. The active level (active high or active low) for the outputs is established with the OUTLVL command. "High" means that current is flowing and no voltage is present at the output terminal; conversely, "low" means that no current is flowing and a voltage may be present at the output terminal. If the active level is set to active low (OUTLVLØ – default), the TOUT response indicates active with a one (1) and inactive with a zero (Ø). If the active level is set to active high (OUTLVL1), the TIN response indicates active with a zero (Ø) and inactive with a one (1).

The general-purpose programmable outputs are returned first, followed by the auxiliary outputs. Output bit assignments vary by product (to ascertain the output bit pattern for your product, refer to page 6 of this document). The outputs are numbered 1 to *n* (*n* depends on the product) from left to right.

If a specific output is required, the bit number can be placed after the TOUT command. For example, TOUT.5 would return bit 5, which corresponds to output 5.

TPANI — Transfer Position of ANI Inputs *(-ANI Option Only)*

		Product	Rev
Type	Transfer	AT6n00	n/a
Syntax	<!><i>TPANI	AT6n50-ANI	3.3
Units	i = analog input number	610n	n/a
Range	1-4 (product dependent)	615n-ANI	3.3
Default	n/a	620n	n/a
Response	TPANI: *TPANI1.963,1.453 1TPANI: *1TPANI1.963	625n-ANI	3.3
See Also	[ANI], ANIPOL, [CA], CMDDIR, [FB], [PANI], PSET, SCALE, SCLD, SFB, TANI, TFB, TCA	6270-ANI	3.0

The TPANI command returns the value of the ANI analog inputs as modified by scaling (SCLD), offset (PSET), polarity (ANIPOL), and commanded direction polarity (CMDDIR).

The TPANI and PANI commands are designed for applications in which the ANI input is scaled and/or used as position feedback. If you are using the ANI input to monitor an analog signal, the TANI and ANI commands would be more appropriate (TANI and ANI values are measured in volts and are unaffected by scaling, offset, polarity, or command direction).

The TPANI value is represented in analog-to-digital converter (ADC) units if scaling is disabled (SCALEØ). The ADC has a 14-bit resolution, giving a range of +8191 to -8192 counts when using the full ±10V range of the ANI input (819 counts/volt). If scaling is enabled (SCALE1), an SCLD scale factor of 819 (the default value) allows units of volts to be reported.

TPC — Transfer Position Commanded

		Product	Rev
Type	Transfer	AT6n00	n/a
Syntax	<!><@><a>TPC	AT6n50	1.0
Units	Reported value represents distance units (scalable)	610n	n/a
Range	Range of the reported value is ±2,147,483,648	615n	1.0
Default	n/a	620n	n/a
Response	TPC: *TPC+0,+0 1TPC: *1TPC+0	625n	1.0
See Also	CMDDIR, ERES, [PC], [PCC], PSET, SCALE, SCLD, SMPER, TAS, TFB, TPCC, TPER	6270	1.0

This command allows you to display the current *commanded position* of each axis. The reported value is measured in encoder, resolver, LDT, or ANI steps and is scaled by the distance scaling factor (SCLD) if scaling is enabled with the SCALE1 command.

The current commanded position is determined by the servo controller's move profile routine. The commanded position profile is the *command* to the servo system that the motor must follow. The *actual position*, displayed with the TFB command, is the position read by the feedback device.

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

The commanded position (TPC) and the actual position (TFB) are used in the control algorithm to calculate the position error (TPC - TFB = TPER) and thereby determine the corrective control signal.

Response for TPC: *TPCr,r where r is the position value (or scaled value)
Response for 1TPC: *1TPCr where r is the position value

Example:

```
TPC             ; Display the current commanded position for each axis:
                ; *TPC4000,4000,4000,4000 (setpoints displayed in steps)
TFB             ; Display the current actual position for each axis:
                ; *TFB4004,4005,4004,4003 (actual positions displayed in steps)
TPER            ; Display current position error of each axis:
                ; *TPER-4,-5,-4,-3 (error displayed in steps)
```

TPCA — Transfer Position of Captured ANI Input

		Product	Rev
Type	Transfer	AT6n00	n/a
Syntax	<!><a>TPCAc	AT6n50-ANI	1.0
Units	c = letter of trigger input	610n	n/a
Range	n/a	615n-ANI	1.0
Default	n/a	620n	n/a
Response	TPCAA: *TPCAA+0,+0,+0,+0 1TPCAA: *1TPCAA+0	625n-ANI	1.0
See Also	[ANI], ANIPOL, [CA], CMDDIR, INFNC, [PANI], [PCA], PSET, SCALE, SCLD, SFB, [SS], SSFR, TANI, TCA, TFB, TPANI, TSS	6270-ANI	1.0

The Transfer Position of Captured ANI (TPCA) command displays the current captured ANI value (counts). After displaying the captured ANI value, the respective position capture status bit (reported with the TSS or SS commands) is cleared, but the position information remains in the register until it is overwritten by a subsequent position capture from the trigger input.

The TPCA value is affected by:

- Scaling (SCLD) if scaling is enabled (SCALE1)
- Position offset (PSET)
- Feedback polarity (ANIPOL)
- Commanded direction polarity (CMDDIR)

The ANI input value can be captured only by a trigger input signal (trigger A, B, C or D). The appropriate trigger inputs must be defined as *trigger interrupt* inputs with the INFNCi-H command, where i is the programmable input bit number representing trigger inputs A, B, C or D, respectively (note that the input bit numbers vary by product — refer to page 6 to verify). Once defined, an active signal on the specified

trigger input will capture the ANI values on all axes. The ANI information is stored in registers and is available at the next 1-ms update through the use of the PCA and TPCA commands.

POSITION CAPTURE ACCURACY

If ANI feedback is selected with the SFB command, the captured ANI value is interpolated from the last sampled ANI input value and rate of change of the ANI input value, and the time elapsed since the last sample. The position sample rate is determined by the SSFR and INDAX commands (*system update rate*). The accuracy of the position capture is ±50μs x velocity.

If ANI feedback is NOT selected with the SFB command, the last sampled ANI value is simply stored as the captured ANI value. The accuracy is one system update period (determined by SSFR and INDAX).

If you issue a PSET (establish absolute position reference) command, any previously captured ANI input values will be offset by the value specified in the PSET command.

Response for TPCAA: *TPCAAr,r,r,r where r is ANI counts (or SCLD scaled value)
Response for 1TPCAA: *1TPCAAr where r is ANI counts (or SCLD scaled value)

TPCC Transfer Captured Commanded Position

		Product	Rev
Type	Transfer		
Syntax	<!><a>TPCCc	AT6n00	n/a
Units	c = letter of trigger input	AT6n50	1.0
Range	n/a	610n	n/a
Default	n/a	615n	1.0
Response	TPCCA: *TPCCA+0,+0,+0,+0 1TPCCA: *1TPCCA+0	620n	n/a
		625n	3.0
See Also	[CA], CMDDIR, INFNC, [PC], [PCA], [PCC], PSET, SCALE, SCLD, SFB, [SS], TCA, TFB, TPC, TSS	6270	1.0

The Transfer Captured Commanded Position (TPCC) command displays the current captured commanded position. After displaying the captured position, the respective position capture status bit (reported with the TSS or SS commands) is cleared, but the position information remains in the register until it is overwritten by a subsequent position capture from the trigger input.

If scaling is enabled (SCALE1), the commanded position is scaled by the distance scaling factor (SCLD). If scaling is NOT enabled (SCALEØ), the value assigned will be actual commanded counts.

The commanded position can be captured only by a trigger input signal (trigger A, B, C or D). The appropriate trigger inputs must be defined as *trigger interrupt* inputs with the INFNCi-H command, where i is the programmable input bit number representing trigger inputs A, B, C or D, respectively (note that the input bit numbers vary by product — refer to page 6 to verify). Once defined, an active signal on the specified trigger input will interpolate the current commanded position for all axes. The captured position is interpolated from the last sampled position (of the feedback device selected with the SFB command), the last sampled position error, and the time elapsed since the last sample. The position sample rate is determined by the SSFR and INDAX commands (*system update rate*). The accuracy of the position capture is ±50μs x velocity.

If you issue a PSET (establish absolute position reference) command, any previously captured commanded positions will be offset by the PSET command value.

Response for TPCCA: *TPCCAr,r,r,r where r is commanded counts (or scaled value)
Response for 1TPCCA: *1TPCCAr where r is commanded counts (or scaled value)

TPCE Transfer Position of Captured Encoder

		Product	Rev
Type	Transfer	AT6n00	2.0
Syntax	<!><a>TPCEc	OEM-AT6n00	n/a
Units	n/a	AT6n50	1.0
Range	n/a	610n	4.0
Default	n/a	615n	1.0
Response	TPCEA: *TPCEA+0,+0,+0,+0 1TPCEA: *1TPCEA+0	620n	2.0
See Also	CMDDIR, ENCPOL, INFNC, [PCE], [PCM], PSET, SCALE, SCLD, SFB, SSFR, TPCM, TPE	625n	1.0
		6270	1.0

The Transfer Position of Captured Encoder (TPCE) command displays the current captured encoder position, from the time of the last trigger interrupt. After displaying the captured encoder position, the respective position capture status bit (reported with the TSS or SS commands) is cleared, but the position information remains in the register until it is overwritten by a subsequent position capture from the trigger input.

The encoder position can be captured only by a trigger input signal (trigger A, B, C or D). The appropriate trigger inputs must be defined as *trigger interrupt* inputs with the INFNCi-H command, where i is the programmable input bit number representing trigger inputs A, B, C or D, respectively (note that the input bit numbers vary by product — refer to page 6 to verify).

Steppers: Once defined, an active trigger input signal from any defined trigger will latch the current encoder positions from all axes and store them in their respective captured encoder arrays. Although the latching may be delayed up to 50 µs from the time the trigger becomes active, all encoder positions are captured within a few microseconds of each other.

If the encoder step mode (ENC1) and scaling (SCALE) are enabled, the value returned is scaled by the distance scaling factor (SCLD). If the encoder step mode or scaling are not enabled, the value returned is actual encoder counts.

Servos: An active trigger input signal from any defined trigger will capture the current encoder positions from all axes. If encoder feedback is selected with the last SFB command, the captured position is interpolated from the last sampled encoder position and velocity, and the time elapsed since the last sample. The position sample rate is determined by the SSFR and INDAX commands (*system update rate*). The accuracy of the position capture is ±50µs x velocity. If encoder feedback is not selected with the SFB command, the last sampled position is simply stored as the captured position, and the accuracy is one system update period (determined by the SSFR and INDAX commands).

Regardless of the SFB selection, one encoder position is latched in hardware within ±1 encoder count (at max. encoder frequency) when its dedicated trigger input is activated (see table below).

Encoder	AT6250	AT6450	615n*	625n	6270	OEM625n
ENCODER 1	TRG-A	TRG-A	TRG-A	TRG-A	TRG-A	TRG-A
ENCODER 2	TRG-B	TRG-B	TRG-B	TRG-B	n/a	TRG-B
ENCODER 3	TRG-C	TRG-C	n/a	TRG-C	n/a	n/a
ENCODER 4	n/a	TRG-D	n/a	n/a	n/a	n/a

*615n: TRG-A captures the internal resolver and TRG-B captures the external encoder.

If scaling is enabled (SCALE1), the value returned is scaled by the distance scaling factor (SCLD). If scaling is not enabled, the value returned is actual encoder counts. AT6250 and 625n: **ENCODER 3** is never scaled.

NOTE: If you issue a PSET (establish absolute position) command, any previously captured encoder positions will be offset by the PSET command value.

Response for TPCEA: *TPCEAr,r,r,r where r is the encoder count (or scaled value)
Response for 1TPCEA: *1TPCEAr where r is the encoder count (or scaled value)

TPCL — Transfer Position of Captured LDT

		Product	Rev
Type	Transfer	AT6n00	n/a
Syntax	<!><a>TPCLc	AT6n50	n/a
Units	c = letter of trigger input	610n	n/a
Range	n/a	615n	n/a
Default	n/a	620n	n/a
Response	TPCLA: *TPCLA+0,+0 1TPCLA: *1TPCLA+0	625n	n/a
See Also	CMDDIR, INFNC, [LDT], LDTPOL, [PCL], PSET, SCALE, SCLD, SFB, [SS], SSFR, TFB, TLDT, TSS	6270	1.0

The Transfer Position of Captured LDT (TPCL) command displays the current captured LDT position. After displaying the captured LDT position, the respective position capture status bit (reported with the TSS or SS commands) is cleared, but the position information remains in the register until it is overwritten by a subsequent position capture from the trigger input.

If scaling is enabled (SCALE1), the position reported is scaled by the distance scaling factor (SCLD). If scaling is not enabled (SCALEØ), the position reported will be actual LDT counts.

The LDT position can be captured only by a trigger input signal (trigger A or B). The appropriate trigger inputs must be defined as *trigger interrupt* inputs with the INFNCi-H command, where i can be 25 or 26, representing trigger inputs A or B, respectively. Once defined, an active signal on the specified trigger input will interpolate the current LDT position from both axes.

POSITION CAPTURE ACCURACY

If LDT feedback is selected with the SFB command, the captured LDT value is interpolated from the last sampled LDT position and velocity, and the time elapsed since the last sample. The position sample rate is determined by the SSFR and INDAX commands (*system update rate*). The accuracy of the position capture is ±50µs x velocity.

If LDT feedback is NOT selected with the SFB command, the last sampled LDT position is simply stored as the captured LDT position. The accuracy is one system update period (determined by SSFR and INDAX).

If you issue a PSET (establish absolute position) command, any previously captured LDT position values will be offset by the PSET command value.

Response for TPCLA: *TPCLAr,r where r is LDT counts (or scaled value)
Response for 1TPCLA: *1TPCLAr where r is LDT counts (or scaled value)

TPCM — Transfer Position of Captured Motor

		Product	Rev
Type	Transfer	AT6n00	2.0
Syntax	<!><a>TPCMc	AT6n50	n/a
Units	n/a	610n	4.0
Range	n/a	615n	n/a
Default	n/a	620n	2.0
Response	TPCMA: *TPCMA+0,+0,+0,+0 1TPCMA: *1TPCMA+0	625n	n/a
See Also	INFNC, [PCE], [PCM], PSET, SCALE, SCLD, [SS], TPCE, TPE, TSS	6270	n/a

The Transfer Position of Captured Motor (TPCM) command returns the current captured motor position, from the time of the last trigger interrupt. After displaying the captured motor position, the respective position capture status bit (reported with the TSS or SS commands) is cleared, but the position information remains in the register until it is overwritten by a subsequent position capture from the trigger input.

The motor position can be captured only by a trigger input signal (trigger A, B, C or D). The appropriate trigger inputs must be defined as *trigger interrupt* inputs with the INFNCi-H command, where i is the programmable input bit number representing trigger inputs A, B, C or D, respectively (note that the input bit numbers vary by product — refer to page 6 to verify). Once defined, an active trigger input signal from any defined trigger will latch the current motor positions from all axes and store them in their respective captured motor arrays. Although the latching may be delayed slightly from the time the trigger becomes active (up to 50 μs), all motor (and encoder) positions are captured within a few microseconds of each other.

NOTE: If you issue a PSET (establish absolute position) command, any previously captured motor positions will be offset by the value specified in the PSET command.

If scaling (SCALE) is enabled, the value returned is scaled by the distance scaling factor (SCLD).

Response for TPCMA: *TPCMAr,r,r,r where r is the motor count
Response for 1TPCMA: *1TPCMAr where r is the motor count

TPE — Transfer Position of Encoder

		Product	Rev
Type	Transfer	AT6n00	1.0
Syntax	<!><a>TPE	OEM-AT6n00	n/a
Units	n/a	AT6n50	1.0
Range	n/a	610n	4.0
Default	n/a	615n	1.0
Response	TPE: *TPE+0,+0,+0,+0 1TPE: *1TPE+0	620n	1.0
See Also	CMDDIR, CNTE, ENCPOL, [FB], [PE], PSET, SCALE, SCLD, SFB, SSFR, TCNT, TFB	625n	1.0
		6270	1.0

The Transfer Position of Encoder (TPE) command returns the current encoder position.

Steppers: The value reported is scaled by the distance scaling factor (SCLD), if encoder step mode (ENC1) and scaling (SCALE) are enabled. If scaling or encoder step mode are not enabled, the value returned is encoder counts. If the encoder channel has been defined as a counter input (CNTE), then the TPE command will report a reading of zero for that specific encoder channel.

Servos: If scaling is enabled (SCALE1), the value returned is scaled by the distance scaling factor (SCLD). If scaling is disabled (SCALEØ), the reported value is the actual position read by the encoder, measured in encoder steps. AT6250 and 625n: **ENCODER 3** is never scaled. 615n: The TPE response regards the internal resolver as axis 1 position, and the external encoder as axis 2 position.

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

Response for TPE: *TPEr,r,r,r where r is the encoder counts (or the scaled value)
Response for 1TPE: *1TPEr where r is the encoder counts (or the scaled value)

TPER — Transfer Position Error

Type	Transfers
Syntax	<!><a>TPER
Units	Reported value represents distance units (scalable)
Range	Range of the reported value is ±2,147,483,648
Default	n/a
Response	TPER: *TPER+0,+0,+0,+0 1TPER: *1TPER+0
See Also	ANIPOL, CMDDIR, DRES, ENCPOL, ERES, [FB], LDTPOL, [PC], [PE], [PER], SFB, SMPER, TANI, TAS, TFB, TLDT, TPE, TPC

Product	Rev
AT6n00	1.0
OEM-AT6n00	n/a
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Transfer Position Error (TPER) command allows you to display the current position error of each axis. The error is displayed in feedback device counts and is scaled by the distance scaling factor (SCLD), if scaling is enabled with the SCALE1 command.

Steppers: This command returns the current position error, and can be used **only** when the encoder mode (ENC1) is enabled.

Servos: The position error is the difference between the commanded position and the actual position read by the feedback device (TPER = TPC - TFB). This error is calculated every sample period and can be displayed at any time using this command.

Response for TPER: *TPERr,r,r,r where r is the error in feedback device counts (or scaled value)
Response for 1TPER: *1TPERr where r is the error in feedback device counts (or scaled value)

Example:

```
TPC             ; Display the current commanded position for each axis:
                ; *TPC4000,4000,4000,4000 (setpoints displayed in steps)
TFB             ; Display the current actual position for each axis:
                ; *TFB4004,4005,4004,4003 (actual positions displayed in steps)
TPER            ; Display current position error of each axis:
                ; *TPER-4,-5,-4,-3 (error displayed in steps)
```

TPM — Transfer Position of Motor

Type	Transfer
Syntax	<!><a>TPM
Units	n/a
Range	n/a
Default	n/a
Response	TPM: *TPM+0,+0,+0,+0 1TPM: *1TPM+0
See Also	[PM], PSET, SCALE, SCLD

Product	Rev
AT6n00	1.0
AT6n50	n/a
610n	4.0
615n	n/a
620n	1.0
625n	n/a
6270	n/a

The Transfer Position of Motor (TPM) command returns the current absolute motor position for all axes. The value returned is scaled by the distance scaling factor (SCLD), if scaling (SCALE) is enabled. If scaling is not enabled, the value returned is motor steps. If in encoder mode (ENC1), then TPM will report back a position of zero for the axes that are in encoder mode (*you do not lose the absolute reference*).

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

Response for TPM: *TPMr,r,r,r where r is the distance value
Response for 1TPM: *1TPMr where r is the distance value

TPMAS — Transfer Current Master Cycle Position

		Product	Rev
Type	Following and Transfer	AT6n00	3.0
Syntax	<!><a>TPMAS	AT6n50	3.0
Units	n/a	610n	4.0
Range	n/a	615n	3.0
Default	n/a	620n	3.0
Response	TPMAS *TPMAS0,0,0,0 1TPMAS *1TPMAS0	625n	3.0
See Also	ENC, FMCLEN, FMCNEW, FMCP, FOLMAS, FOLMD, [FS], [NMCY], [PMAS], SCALE, SCLMAS, TFS	6270	3.0

The TPMAS command transfers the current position of the master within its current master cycle. **The master must be assigned first (**FOLMAS **command) before this command will be useful.**

TPMAS is unique among position transfers, because master cycle position rolls over to zero each time the entire master cycle length (FMCLEN value) has been traveled.

If scaling is enabled (SCALE1), the value returned is scaled by the master scaling factor (SCLMAS). If scaling is disabled (SCALEØ), the value returned is in motor steps is in counts.

For a complete discussion of master cycles, please refer to the Following chapter in the *6000 Series Programmer's Guide.*

TPROG — Transfer Program

		Product	Rev
Type	Transfer	AT6n00	1.0
Syntax	<!>TPROG<t>	AT6n50	1.0
Units	t = text (name of program)	610n	4.0
Range	Text name of 6 characters or less	615n	1.0
Default	n/a	620n	1.0
Response	n/a	625n	1.0
See Also	DEF, TDIR, TMEM	6270	1.0

The Transfer Program (TPROG) command uploads the program specified. If there is no such program, then the error message *INVALID DATA will be generated. To see which programs have been created, use the TDIR command.

TPSHF — Transfer Net Position Shift

		Product	Rev
Type	Following and Transfer	AT6n00	3.0
Syntax	<!><a>TPSHF	AT6n50	3.0
Units	n/a	610n	4.0
Range	n/a	615n	3.0
Default	n/a	620n	3.0
Response	TPSHF *TPSHF0	625n	3.0
See Also	ENC, FMCNEW, FMCP, FOLEN, FSHFC, FSHFD, [PSHF], SCALE, SCLD	6270	3.0

The TPSHF command transfers the net (absolute) slave axis position shift that has occurred since that last FOLEN1 command. The position returned will be the sum of all shifts performed on that axis, or axes, including decelerations due to limits, kill, or stop. The shift value is set to zero each time a new FOLEN1 command or a FOLMAS command (with a value other than zero) is issued.

Steppers: If scaling in enabled (SCALE1), the value returned is scaled by the distance scaling factor (SCLD). If scaling is not enabled, the value returned is in motor steps if in motor step mode (ENCØ) or in encoder steps if in encoder step mode (ENC1).

Servos: If scaling in enabled (SCALE1), the value returned is scaled by the distance scaling factor (SCLD). If scaling is not enabled, the value returned is commanded counts.

TPSLV — Transfer Current Commanded Position of Slave

Type	Following and Transfer
Syntax	<!><a>TPSLV
Units	n/a
Range	n/a
Default	n/a
Response	TPSLV *TPSLV0
See Also	ENC, FMCNEW, FMCP, [PSLV], SCALE, SCLD

Product	Rev
AT6n00	3.0
AT6n50	3.0
610n	4.0
615n	3.0
620n	3.0
625n	3.0
6270	3.0

The TPSLV command transfers the current commanded position of the slave axis. **The master must be assigned first (**FOLMAS **command) before this command will be useful.**

Steppers: If scaling in enabled (SCALE1), the value returned is scaled by the distance scaling factor (SCLD). If scaling is not enabled, the value returned is in motor steps.

Servos: If scaling is enabled (SCALE1), the value returned is scaled by the distance scaling factor (SCLD). If scaling is not enabled, the value returned is commanded counts.

TRACE — Program Trace Mode Enable

Type	Program Debug Tool
Syntax	<!>TRACE<b>
Units	n/a
Range	b = 0 (disable), 1 (enable) or X (don't care)
Default	0
Response	TRACE: *TRACE0
See Also	[,], [#], PORT, [SS], STEP, TRANS, TSS

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Program Trace Mode Enable (TRACE) command enables program trace mode. When in program trace mode, all commands executed are placed in the bus-based product's output data buffer, or transferred out the RS-232 port for stand-alone products, along with the program from which the command came.

Example:

```
DEF pick          ; Begin definition of program named pick
GO1100            ; Initiate motion on axes 1 and 2
IF(VAR1=5)        ; If variable 1 = 5 then do commands between IF and NIF
  GOTOpick1       ; Goto label pick1
  ELSE            ; Else part of IF command
  GOTOpick2       ; Goto label pick2
NIF               ; End IF command
$pick1            ; Label declaration for pick1
GO0011            ; Initiate motion on axes 3 and 4
BREAK             ; Break out of current subroutine or program
$pick2            ; Label declaration for pick2
GO1001            ; Initiate motion on axes 1 and 4
END               ; End program definition
TRACE1            ; Enable trace mode.
VAR1=5            ; Set variable 1 to 5
@LH0              ; Disable all limits
EOT13,10,0        ; Set End-of-Transmission characters to a carriage return
                  ; and a line feed
RUN pick          ; Initiate program pick
```

After executing RUN pick, the following information will be placed in the output buffer, due to the trace mode being enabled. (Assume variable 1 = 5)

```
*PROGRAM=PICK COMMAND=GO1100
*PROGRAM=PICK COMMAND=IF(VAR1=5.0)
*PROGRAM=PICK COMMAND=GOTO PICK1
*PROGRAM=PICK COMMAND=$PICK1
*PROGRAM=PICK COMMAND=GO0011
*PROGRAM=PICK COMMAND=BREAK
```

TRANS — Translation Mode Enable

Type	Program Debug Tool
Syntax	<!>TRANS<b>
Units	n/a
Range	b = 0 (disable), 1 (enable) or X (don't care)
Default	0
Response	TRANS: *TRANS0
See Also	[#], [SS], STEP, TSS

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Translation Mode Enable (TRANS) command enables the program translation mode, in which all commands processed by the 6000 Series product are echoed back in their binary format (hex representation of the binary equivalent), and are not executed. The first byte (first two characters) of the response represents the command's memory requirement. The remaining bytes represent the actual command.

Example:

```
TRANS1          ; Enable translation mode
A10,20,1,1      ; Translate acceleration command A10,20,1,1. Response displayed
                ; is: 13 01 00 00 01 86 A0 00 03 0D 40 00 00 27 10 00 00 27 10.
                ; Note that 13 hex represents a command memory requirement of 19
                ; bytes.
GO1100          ; Translate initiate motion command GO1100. Response displayed
                ; is: 07 07 03 01 01 00 00.   Note that 07 hex represents a
                ; command memory requirement of 7 bytes.
GO0011          ; Translate initiate motion command GO0011. Response displayed
                ; is: 07 07 03 00 00 01 01.   Note that 07 hex represents a
                ; command memory requirement of 7 bytes.
```

TREV — Transfer Revision Level

Type	Transfer
Syntax	<!>TREV
Units	n/a
Range	n/a
Default	n/a
Response	TREV: *TREV92-012163-01-1.0 (response varies by product)
See Also	None

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Transfer Revision Level (TREV) command provides the current revision of the operating system software. It also reports any options, such as contouring, that have been installed. Options can be ordered through your local ATC or distributor.

TRGFN Trigger Functions

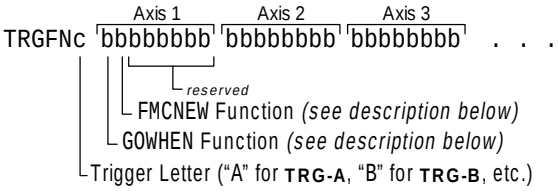
		Product	Rev
Type	Inputs; Following; Motion	AT6n00	3.0
Syntax	<!><@><a>TRGFNcbbbbbbbb bbbbbbbb bbbbbbbb bbbbbbbb	AT6n50	3.0
Units	c = trigger input letter; b = enable bit (see function list below)	610n	4.0
Range	c = A - D; b = 0 (disable), 1 (enable), or X (don't change)	615n	3.0
Default	c = A; b = 0	620n	3.0
Response	TRGFN *TRGFNA00000000_00000000_00000000_00000000 1TRGFN *1TRGFNA00000000	625n	3.0
See Also	[AS], ERROR, ERRORP, FMCNEW, GOWHEN, INDEB, INFNC, [SS], TAS, TSS	6270	3.0

Use the TRGFN command to assign certain command functions to the trigger inputs (**TRG-A** through **TRG-D**). *Note that the number of trigger inputs available varies by product.*

NOTE

The trigger input used in this command must first be defined as a *Trigger Interrupt* input with the INFNCi-H command.

In the TRGFN command syntax, each field of 8 enable bits is for one axis. The "c" in the first data field is for specifying the trigger input (**TRG-A** through **TRG-D**). There are two possible functions, corresponding to the first 2 enable bits in the syntax (the other 6 enable bits per axis are reserved):



TRGFNc1xxxxxxx = GOWHEN function: Suspends execution of the next move until the specified trigger input (c) goes active. If you need execution to be triggered by other factors (e.g., master position, encoder position, etc.) use the GOWHEN command. Refer to the GOWHEN command description for additional details. AS and TAS bit #26 is set when there is a pending GOWHEN condition initiated by a TRGFNc1xxxxxxx command; this bit is cleared when the trigger is activated or when a stop or kill command is issued.

TRGFNcx1xxxxxx = FMCNEW function: Allows a new master cycle to begin when the specified trigger input (c) goes active. Refer to the FMCNEW command description of additional details.

These trigger functions are cleared once the function is complete. To use the trigger to perform a GOWHEN function again, the TRGFN command must be given again.

TRGFN in Compiled Motion: When used in a compiled program, a TRGFNc1xxxxxxx (GOWHEN function) command will pause the profile in progress (motion continues at constant velocity) until the trigger is activated to execute the next move profile. When used in a compiled profile, the TRGFN command consumes one segment of compiled memory. When used in a compiled Following profile, the TRGFN command is ignored on the reverse Following profile (i.e., when the master is moving in the opposite direction of that specified in the FOLMAS command).

Example: (refer also to the FOLEN examples)

```
TRGFNBx1xxxxxx 1  ; When trigger B (TRG-B) goes active, axis 1 will begin a
                  ; new master cycle and axis 2 will execute the move
                  ; commanded with the GO command.
GO01              ; The move on axis 2 is commanded, but will not execute until
                  ; TRG-B becomes active.
```

TSEG — Transfer Number of Free Segment Buffers

Type	Compiled Motion; Transfer
Syntax	<!>TSEG
Units	n/a
Range	n/a
Default	n/a
Response	TSEG: *TSEG258
See Also	MEMORY, TDIR, TMEM, [SEG], [SS], TSS, TSSF

Product	Rev
AT6n00	4.1
AT6n50	4.1
610n	4.0
615n	4.1
620n	4.1
625n	4.1
6270	4.1

The Transfer Number of Free Segment Buffers (TSEG) command returns the number of free segment buffers in compiled memory.

System status bit (see TSSF, TSS, and SS) 29 to set when the compiled memory is 75% full, and bit 30 is set if the compiled memory is 100% full.

TSGSET — Transfer Servo Gain Set

Type	Transfer
Syntax	<!>TSGSETi
Units	i = gain set identification number (see SGSET command)
Range	1 - 5
Default	n/a
Response	(see examples below)
See Also	SFB, SGAF, SGAFN, SGENB, SGI, SGILIM, SGIN, SGP, SGPN, SGSET, SGV, SGVF, SGVFN, SGVN, SOFFS, SOFFSN, SSWG, TGAIN

Product	Rev
AT6n00	n/a
AT6n50	1.0
610n	n/a
615n	1.0
620n	n/a
625n	1.0
6270	1.0

This command allows you to display any of the 5 gain sets that you saved with the SGSET command. Up to 5 gain sets can be saved.

NOTE

The tuning gains in a given gain set are specific to the feedback source that was in use (selected with the last SFB command) at the time the gains were established with the respective gain commands (SGI, SGP, etc.).

Example:

```
SGP5,5,10,10    ; Set the gain for the proportional gain
SGI.1,.1,0,0    ; Set the gain for the integral gain
SGV50,60,0,0    ; Set the gain for the velocity gain
SGVF5,6,10,11   ; Set the gain for the velocity feedforward gain
SGAF0,0,0,0     ; Set the gain for the acceleration feedforward gain
SGSET3          ; Assign the SGP, SGI, SGV, SGVF, & SGAF gains to servo gain set 3
SGP75,75,40,40  ; Set the gain for the proportional gain
SGI5,5,5,7      ; Set the gain for the integral gain
SGV1,.45,2,2    ; Set the gain for the velocity gain
SGVF0,8,0,9     ; Set the gain for the velocity feedforward gain
SGAF18,20,22,24 ; Set the gain for the acceleration feedforward gain
SGSET1          ; Assign the SGP, SGI, SGV, SGVF, & SGAF gains to servo gain set 1
SGENB1,3,3,1    ; Enable gain set 1 on axis 1 & 4 and enables gain set 3 on
                ; axis 2 & 3
TSGSET1         ; Display gain set 1.  Response should be:
                ;   *SGP75,75,40,40
                ;   *SGI5,5,5,7
                ;   *SGV1,.45,2,2
                ;   *SGVF0,8,0,9
                ;   *SGAF18,20,22,24
TSGSET3         ; Display gain set 3.  Response should be:
                ;   *SGP5,5,10,10
                ;   *SGI.1,.1,0,0
                ;   *SGV50,60,0,0
                ;   *SGVF5,6,10,11
                ;   *SGAF0,0,0,0
```

TSS Transfer System Status

		Product	Rev
Type	Transfer	AT6n00	2.0
Syntax	<!>TSS<.i>	AT6n50	1.0
Units	i = system status bit number	610n	4.0
Range	1 - 32	615n	1.0
Default	n/a	620n	2.0
Response	TSS: *TSS1000_1000_0000_0000_0000_0000_0000_0000 TSS.1: *1 (status of status bit #1—system is ready)	625n	1.0
See Also	PORT, [SS], TAS, TCMDER, TRGFN, TSSF, TSTAT	6270	1.0

The Transfer System Status (TSS) command provides information on the 32 system status bits.

Response for TSS (b can equal Ø, 1, X, or x): *TSSbbbb_bbbb_bbbb_bbbb_bbbb_bbbb_bbbb_bbbb

Bit #1 ^ Bit #32 ^

FULL-TEXT STATUS REPORT AVAILABLE

The TSS status command reports a binary bit report. If you would like to see a more descriptive text-based report, use the TSSF command description.

BIT (Left to Right)	**Function** (1 = yes, Ø = no)	**BIT** (Left to Right)	**Function** (1 = yes, Ø = no)
1	System Ready	17	Loading Thumbwheel Data ([TW])
2	Reserved	18	External Program Select Mode (INSELP)
3	Executing a Program	19	Dwell in Progress (T command)
4	Immediate Command (set if last command was immediate)	20	Waiting for RP240 Data—[DREAD] or [DREADF] (stand-alone products)
5	In ASCII Mode	21	RP240 Connected (stand-alone products) — current PORT setting only
6	In Echo Mode (stand-alone products) — current PORT setting only	22	Non-volatile Memory Error (stand-alone products)
7	Defining a Program	23	Servo data gathering transmission in progress (servo products)
8	In Trace Mode	24	Reserved
9	In Step Mode	25 *	Position captured with **TRG-A**
10	In Translation Mode (must use fast status area to see—bus-based)	26 *	Position captured with **TRG-B**
11	Command Error Occurred (bit is cleared when TCMDER is issued)	27 *	Position captured with **TRG-C**
12	Break Point Active (BP)	28 *	Position captured with **TRG-D**
13	Pause Active	29	Compiled memory is 75% full
14	Wait Active (WAIT)	30	Compiled memory is 100% full
15	Monitoring On Condition (ONCOND)	31 **	Compile operation failed (PCOMP) **
16	Waiting for Data (READ)	32	Reserved

* Bits 25 through 28 are cleared when the respective captured position is read with the [CA], [PCA], [PCC], [PCE], [PCL], [PCM], TCA, TPCA, TPCC, TPCE, TPCL, or TPCM commands, but the position information is still available from the respective register until it is overwritten by a subsequent position capture.

** Bit #31: failed PCOMP compile is cleared on power up, RESET, or after successful compile. Possible causes include:

- Errors in profile design (e.g., change direction while at non-zero velocity; distance & velocity equate to < 1 count per system update; preset move profile ends in non-zero velocity)
- Profile will cause a Following error (see TFSF, TFS, or FS command descriptions)
- Out of memory (see TSS bit #30)
- Axis already in motion at the time of the PCOMP command
- Loop programming errors (e.g., no matching PLOOP or PLN; more than 4 embedded PLOOP/END loops)

TSSF — Transfer System Status (full-text report)

Type	Transfer
Syntax	<!>TSSF
Units	n/a
Range	n/a
Default	n/a
Response	TSSF: (see example below)
See Also	PORT, [SS], TAS, TCMDER, TRGFN, TSS, TSTAT

Product	Rev
AT6n00	2.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	2.0
625n	1.0
6270	1.0

The TSSF command returns a text-based status report of all axes. This is an alternative to the binary report (TSS).

Example response (for 610n):

```
*TSSF
*System Ready        YES     Thumbwhl Data Load  NO
*Reserved            NO      Ext Prog Sel Mode   NO
*Program Executing   NO      Time Command        NO
*Immediate Comm Last NO      Waiting RP240 Data  NO
*
*Ascii Mode          YES     RP240 Connected     NO
*Echo Mode           YES     Memory Error        NO
*Defining a Program  NO      Servo Data Transfer NO
*Trace Mode          NO      Reserved            NO
*
*Step Mode           NO      Pos captured TRG-A  NO
*FS Translate Mode   NO      Pos captured TRG-B  NO
*Command Error       YES     Pos captured TRG-C  NO
*Break Point Active  NO      Pos captured TRG-D  NO
*
*Pause Active        NO      Comp Mem Near Full  NO
*Wait Active         NO      Compiled Mem Full   NO
*Checking On Conds   NO      Compile Failed      NO
*Waiting for Data    NO      Reserved            NO
```

TSTAT — Transfer Statistics

Type	Transfer
Syntax	<!>TSTAT
Units	n/a
Range	n/a
Default	n/a
Response	TSTAT: (See below)
See Also	TANV, TAS, TCNT, TDIR, TEX, TFB, TIN, TINT, TLABEL, TLIM, TMEM, TOUT, TPC, TPCE, TPE, TPM, TPROG, TREV, TSS, TTIM, TUS, TVEL

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The following is an example (for the AT6400) of information provided by the Transfer Statistics (TSTAT) command (**NOTE:** The response for each 6000 Series product will vary slightly.):

```
*AT6400 Revision 92-012163-01-1.0
*Auxiliary Board Type 1 Participating Axes 4
*Current Motor Position +0,+0,+0,+0
*Hard Limits Enabled 3,3,3,3 Soft Limits Enabled 0,0,0,0
*0 Programs Defined; Scale Enabled 0; Inputs Enable 0; Outputs Enable 0
*Drive Resolution 25000,25000,25000,25000
*Encoder Resolution 4000,4000,4000,4000
*Acceleration Scaler SCLA25000,25000,25000,25000
*Distance Scaler SCLD1,1,1,1
*Velocity Scaler SCLV25000,25000,25000,25000
*Acceleration A10.0000,10.0000,10.0000,10.0000
*Deceleration AD10.0000,10.0000,10.0000,10.0000
*Distance D+25000,+25000,+25000,+25000
*Velocity V1.0000,1.0000,1.0000,1.0000
*Input Configuration AAAA_BBBB_CCCC_AAAA_BBBB_CCCC_AAAA
*Input State   1111_0000_1111_0000_1111_0000_1111
*Output Configuration AAAA_BBBB_CCCC_AAAA_BBBB_CCCC
*Output State  1111_0000_1111_0000_1111_0000
*Interrupt Bits Enabled 1111_0000_1111_0000_1111_0000_1111_000
*System Status 1111_0000_1111_0000_1111_0000_1111_0000
*Axis#1 Status 1111_0000_1111_0000_1111_0000_1111_0000
*Axis#2 Status 1111_0000_1111_0000_1111_0000_1111_0000
*Axis#3 Status 1111_0000_1111_0000_1111_0000_1111_0000
*Axis#4 Status 1111_0000_1111_0000_1111_0000_1111_0000
```

TSTLT — Transfer Settling Time

		Product	Rev
Type	Transfer		
Syntax	<!><a>TSTLT	AT6n00	4.1
Units	Reported value represents milliseconds	AT6n50	1.0
Range	n/a	610n	4.0
Default	n/a	615n	1.0
Response	TSTLT: *TSTLT502,483 1TSTLT: *1TSTLT502	620n	4.1
		625n	1.0
See Also	STRGTD, STRGTE, STRGTT, STRGTV	6270	1.0

TSTLT allows you to display the actual time it took the last move to settle into the target zone (that is, within the distance zone defined by STRGTD and less than or equal to the velocity defined by STRGTV). The reported value represents milliseconds. **This command is usable whether or not the Target Zone Settling Mode is enabled with the STRGTE command.**

Stepper Products must use an encoder and be in the encoder step mode (ENC1) to use this command.

*** For a more information on target zone operation, refer to the *6000 Series Programmer's Guide*.

TTIM — Transfer Timer

		Product	Rev
Type	Transfer		
Syntax	<!>TTIM	AT6n00	1.0
Units	Reported value represents milliseconds	AT6n50	1.0
Range	Maximum count is 999,999,999 (approx. 11 days, 13 hours)	610n	4.0
Default	n/a	615n	1.0
Response	TTIM: *TTIM64000	620n	1.0
See Also	T, [TIM], TIMINT, TIMST, TIMSTP	625n	1.0
		6270	1.0

The Transfer Timer (TTIM) command returns the current value of the timer in milliseconds. The timer is started with the TIMST command, and stopped with the TIMSTP command.

TUS — Transfer User Status

		Product	Rev
Type	Transfer		
Syntax	<!>TUS<.i>	AT6n00	1.0
Units	i = user status bit number	AT6n50	1.0
Range	1 - 16	610n	4.0
Default	n/a	615n	1.0
Response	TUS: *TUS1111_0000_1111_0000 TUS.4: *1 (user status bit 4 is reported)	620n	1.0
		625n	1.0
See Also	INDUSE, INDUST, [US]	6270	1.0

The Transfer User Status (TUS) command returns the current bit pattern for the user status word. All 16 bits of the user status word are defined with the INDUST command. Each bit can correspond to an axis status bit, a system status bit, an input, or an interrupt bit.

Example:

```
INDUSE1          ; Enable the use of INDUST command
INDUST1-5A       ; User status bit 1 defined as axis 1 status bit 5
INDUST2-3D       ; User status bit 2 defined as axis 4 status bit 3
INDUST3-5J       ; User status bit 3 defined as input 5
INDUST4-1K       ; User status bit 4 defined as interrupt status bit 1
INDUST16-2I      ; User status bit 16 defined as system status bit 2
TUS              ; Return the state of the user status word
```

TVEL — Transfer Current Commanded Velocity

		Product	Rev
Type	Transfer	AT6n00	1.0
Syntax	<!><a>TVEL	AT6n50	1.0
Units	Reported value is in units/sec	610n	4.0
Range	n/a	615n	1.0
Default	n/a	620n	1.0
Response	TVEL: *TVEL23.3450,23.0000,45.7800,456.7800 1TVEL: *1TVEL23.3450	625n	1.0
See Also	ERES, LDTRES, SCALE, SCLV, TVELA, V, [VEL]	6270	1.0

Steppers: The Transfer Current Velocity (TVEL) command returns the current motor velocity, regardless of the setting for the Encoder/Motor Step Mode (ENC) command. It does not return the programmed velocity (V). The value returned will be scaled by the velocity scaling factor (SCLV), if scaling is enabled (SCALE1). If scaling has not been enabled, the value returned will be in revolutions/sec (actual velocity in steps/sec divided by the drive resolution DRES value). The TVEL command does **not** reflect the actual velocity during feedrate override (FR).

Servos: The value reported is the current commanded velocity as calculated by the DSP's move profile routine; it is not necessarily the velocity programmed with the V command. The value reported will be scaled by the velocity scaling factor (SCLV), if scaling is enabled (SCALE1). If scaling is not enabled, the value returned will be in encoder or resolver revs/sec, LDT inches/sec, or ANI volts/sec.

TVELA — Transfer Current Actual Velocity

		Product	Rev
Type	Transfer	AT6n00	4.1
Syntax	<!><a>TVELA	OEM-AT6n00	n/a
Units	Reported value is in units/sec	AT6n50	1.0
Range	n/a	610n	4.0
Default	n/a	615n	1.0
Response	TVELA: *TVELA+1.55,-3.25,-5.55,+2.30 1TVELA: *TVELA+1.55	620n	4.1
See Also	SCALE, SCLV, SFB, TVEL, V, [VEL], [VELA]	625n	1.0
		6270	1.0

The Transfer Current Actual Velocity (TVELA) command reports the current velocity as derived from the feedback device. The sign determines the direction of motion. You can use the TVELA command at all times; therefore, even if no motion is being commanded, TVELA will still report a non-zero value as it detects the servoing action.

Units of Measure:

Steppers: The velocity is always revs/sec (actual velocity in steps/sec multiplied by the ERES value).
Servos: If scaling is enabled (SCALE1), the velocity value will be scaled by the velocity scaling factor (SCLV). If scaling is not enabled (SCALEØ), the value returned will be in encoder or resolver revs/sec, LDT inches/sec, or ANI volts/sec.

Example:

```
TVELA           ; Reports the current actual velocity; since no motion is
                ; commanded, the servoing velocities are reported.
                ; Example response is:  *TVELA+0.0097,-0.0027,+0.0103,-0.0044
```

TVMAS — Transfer Current Master Velocity

Type	Following and Transfer
Syntax	<!><a>TVMAS
Units	n/a
Range	n/a
Default	n/a
Response	TVMAS *TVMAS0,0,0,0 1TVMAS *1TVMAS0
See Also	ENC, FFILT, FOLMAS, SCALE, SCLMAS, V, [VMAS]

Product	Rev
AT6n00	3.0
AT6n50	3.0
610n	4.0
615n	3.0
620n	3.0
625n	3.0
6270	3.0

The TVMAS command transfers the current velocity of the master. **The master must be assigned first (**FOLMAS **command) before this command will be useful.**

The precision of the reported TVMAS value is dependent upon the FFILT filter value (details are provided in the "Master Position Filtering" section in the Following chapter of the *6000 Series Programmer's Guide*.

If scaling is enabled (SCALE1), the value returned is scaled by the master scaling factor (SCLMAS). If scaling is disabled (SCALE0), the value returned is in counts/sec.

[TW] — Thumbwheel Assignment

Type	Assignment or Comparison
Syntax	TWi (See below for examples)
Units	i = sets used by INPLC, INSTW, OUTPLC and OUTTW
Range	1 - 8
Default	n/a
Response	n/a
See Also	INPLC, INSTW, OUTPLC, OUTTW, [SS], TSS

Product	Rev
AT6n00	1.0
OEM-AT6n00	3.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Thumbwheel Assignment (TW) command, executed from within another command, reads data from a parallel device and loads it into the command field the TW command is occupying. Rule of Thumb for command value substitutions: If the command syntax shows that the command field requires a real number (denoted by <r>) or and integer value (denoted by <i>), you can use the TW substitution (e.g., V2,(TW)).

The value of the TW command designates which input and output set to use. TW values 1-4 correspond to INSTW and OUTTW sets 1 - 4, respectively. TW values 5-8 correspond to INPLC and OUTPLC sets 1 - 4, respectively.

The TW command can be used as a variable assignment (VAR1=TW2) or in another command (e.g., A10,(TW2),10,1). However, the TW command cannot be used in an expression such as VAR4=1 + TW2 or IF(TW2<8).

TW1 through TW4 are designed to interface with Compumotor's TM8 Thumbwheel Module. The outputs, specified by OUTTW, strobe data in a binary pattern and data is read one digit per access. TW5 through TW8 are designed to interface with PLCs or passive thumbwheel devices. The outputs, specified by OUTPLC, strobe data one at a time and data is read two digits per access.

For more information on interfacing thumbwheels, refer to your product's *Installation Guide*.

Example:

```
INSTW2,1-4,5      ; Set INSTW set 2 as BCD digits on inputs 1 - 4, with
                  ; input 5 as the sign bit
OUTTW2,1-3,4,50   ; Set OUTTW set 2 as output strobes on outputs 1 - 3,
                  ; with output 4 as the output enable bit, and strobe time
                  ; of 50 milliseconds
A(TW2)            ; Read data into axis 1 acceleration using INSTW set 2
                  ; and OUTTW set 2 as the data configuration
```

UNTIL( ) Until Part of Repeat Statement

Type	Program Flow Control
Syntax	<!>UNTIL(expression)
Units	n/a
Range	Up to 80 characters (including parentheses)
Default	n/a
Response	n/a
See Also	JUMP, REPEAT

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Until Part of Repeat Statement (UNTIL( )) command, in conjunction with the REPEAT command, provide a means of conditional program flow. The REPEAT command marks the beginning of the conditional statement. The commands between the REPEAT and the UNTIL command are executed at least once. Upon reaching the UNTIL command, the expression contained within the UNTIL command is evaluated. If the expression is false, the program flow is redirected to the first command after the REPEAT command. If the expression is true, the first command after the UNTIL command is executed.

Up to 16 levels of REPEAT ... UNTIL( ) commands may be nested.

NOTE: Be careful about performing a GOTO between REPEAT and UNTIL. Branching to a different location within the same program will cause the next REPEAT statement encountered to be nested within the previous REPEAT statement, unless an UNTIL command has already been encountered. The JUMP command should be used in this case.

All logical operators (AND, OR, NOT), and all relational operators (=, >, >=, <, <=, <>) can be used within the UNTIL expression. There is no limit on the number of logical operators, or on the number of relational operators allowed within a single UNTIL expression.

The limiting factor for the UNTIL expression is the command length. **The total character count for the UNTIL command and expression cannot exceed 80 characters.** For example, if you add all the letters in the UNTIL command and the letters within the ( ) expression, including the parentheses and excluding the spaces, this count must be less than or equal to 80.

All assignment operators (A, AD, ANI, ANV, AS, CNT, D, DAC, DPTR, ER, IN, INO, LIM, MOV, OUT, PC, PCA, PCC, PCE, PCL, PCM, PE, PER, PM, SS, TIM, US, V, VEL, etc.) can be used within the UNTIL expression.

Example:

```
REPEAT          ; Beginning of REPEAT ... UNTIL( ) loop
GO1110          ; Initiate motion on axes 1, 2, and 3
IF(IN=b1X0)     ; Specify IF condition to be input 1 = 1, input 3 = Ø
VAR1=VAR1+1     ; If condition comes true increment variable 1 by 1
ELSE            ; Else part of IF condition
TPE             ; If condition does not come true transfer position of
                ; all encoders
NIF             ; End IF statement
UNTIL(VAR1=12)  ; Repeat loop until variable 1 = 12
```

[US] User Status

Type	Assignment or Comparison
Syntax	See below
Units	n/a
Range	n/a
Default	n/a
Response	n/a
See Also	INDUSE, INDUST, TUS

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

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

Syntax: VARBn=US where n is the binary variable number,
or [US] can be used in an expression such as IF(US=b11Ø1), or IF(US=h7)

All 16 bits of the user status word are defined with the INDUST command. Each bit can correspond to an axis status bit, a system status bit, an input, or an interrupt bit.

If it is desired to assign only one bit of the user status value to a binary variable, instead of all 16, the bit select (.) operator can be used. For example, VARB1=US.12 assigns user status bit 12 to binary variable 1.

Example:

```
VARB1=US           ; User status assigned to binary variable 1
VARB2=US.12        ; User status bit 12 assigned to binary variable 2
VARB2              ; Response, if bit 12 is set to 1, will be:
                   ;    *VARB2=XXXX_XXXX_XXX1_XXXX_XXXX_XXXX_XXXX_XXXX
IF(US=b111011X11)  ; If the user status contains 1's in bit locations
                   ; 1, 2, 3, 5, 6, 8, and 9, and a 0 in bit location 4,
                   ; do the IF statement
TREV               ; Transfer revision level
ELSE               ; Else
IF(US=h7F00)       ; If the user status contains 1's in bit locations
                   ; 1, 2, 3, 5, 6, 7, and 8, and 0's in every other bit
                   ; location, do the IF statement
TSTAT              ; Transfer statistics
NIF                ; End of second if statement
NIF                ; End of first IF statement
```

V — Velocity

Type	Motion
Syntax	<!><@><a>V<r>,<r>,<r>,<r>
Units	r = units/sec
Range	Steppers: 0.00000-1600000 (max. depends on SCLV and PULSE) Servos: 0.00000-1600000 (max. depends on SCLV)
Default	1.0000
Response	V: *V1.0000,1.0000,1.0000,1.0000 1V: *1V1.0000
See Also	GO, MC, PULSE, SCALE, SCLV, SSV, TSTAT, TVEL, TVELA, [V], [VEL], [VELA], VF

Product	Rev
AT6n00	1.0
AT6n50	3.1
610n	4.0
615n	3.0
620n	1.0
625n	3.1
6270	3.0

The Velocity (V) command defines the speed at which the motor will run when given a GO command. The motor will accelerate at a predefined acceleration (A) rate, before reaching the velocity (V) specified. The maximum velocity attainable is 1,600,000 units/sec.

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

Steppers: The entered value is always in reference to motor steps, not encoder steps, regardless of the ENC command setting. If scaling is not enabled (SCALEØ), the velocity value is entered in motor revs/sec; this value is internally multiplied by the drive resolution (DRES) value to obtain a velocity value in motor steps/sec for the motor trajectory calculations. If scaling is enabled (SCALE1), the entered velocity value is internally multiplied by the velocity scaling factor (SCLV) to convert user units/sec to motor steps/sec.

Servos: If scaling is not enabled (SCALEØ), the velocity value is entered in encoder or resolver revs/sec, LDT inches/sec, or ANI volts/sec; encoder/resolver and LDT values are internally multiplied by the encoder/resolver resolution (ERES) value or the LDT resolution (LDTRES) value to obtain a velocity value in steps/sec for the motion trajectory calculations. If scaling is enabled (SCALE1), the entered velocity value is internally multiplied by the velocity scaling factor (SCLV) to convert user units/sec to encoder, resolver, LDT, or ANI steps/sec.

ON-THE-FLY CHANGES: While running in the continuous mode (MC1), you can change velocity *on the fly* (while motion is in progress) in two ways. One way is to send an immediate velocity command (!V) followed by an immediate go command (!GO). The other, and more common, way is to enable the continuous command execution mode (COMEXC1) and execute a buffered velocity command (V) followed by a buffered go command (GO).

Example:

```
MA0000                ; Incremental index mode for axes 1-4
MC0000                ; Preset index mode for axes 1-4
SCALE1                ; Enable scaling
SCLA25000,25000,1,1   ; Set the acceleration scaling factor for axes 1 and 2 to
                      ; 25000 steps/unit/unit, axes 3 and 4 to 1 step/unit/unit
SCLV25000,25000,1,1   ; Set the velocity scaling factor for axes 1 and 2 to
                      ; 25000 steps/unit, axes 3 and 4 to 1 step/unit
SCLD1,1,1,1           ; Set the distance scaling factor for axes 1, 2, 3, and 4
                      ; to 1 step/unit
A10,12,1,2            ; Set the acceleration to 10, 12, 1, and 2 units/sec/sec
                      ; for axes 1, 2, 3 and 4
V1,1,1,2              ; Set the velocity to 1, 1, 1, and 2 units/sec for
                      ; axes 1, 2, 3 and 4
D100000,1000,10,100   ; Set the distance to 100000, 1000, 10, and 100 units
                      ; for axes 1, 2, 3 and 4
GO1100                ; Initiate motion on axes 1 and 2, 3 and 4 do not move
```

[V] Velocity (Programmed) Assignment

Type	Assignment or Comparison
Syntax	See below
Units	n/a
Range	n/a
Default	n/a
Response	n/a
See Also	GO, SCALE, SCLV, SSV, V, [VEL]

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

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

Syntax: VARn=aV where n is the variable number, and a is the axis number, or [V] can be used in an expression such as IF(1V<25)

When assigning the velocity value to a variable, an axis specifier must always precede the assignment (V) command or it will default to axis 1 (e.g., VAR1=1V). When making a comparison to the programmed velocity, an axis specifier must also be used (e.g., IF(1V<2Ø)). The (V) value used in any comparison, or in any assignment statement is the programmed (V) value. If the actual velocity information is required, refer to the VEL command.

Steppers: The value is always in reference to motor steps, not encoder steps, regardless of the ENC command setting. If scaling is not enabled (SCALEØ), the velocity value represents motor revs/sec; this value is internally multiplied by the drive resolution (DRES) value to obtain a velocity value in motor steps/sec for the motor trajectory calculations. If scaling is enabled (SCALE1), the velocity value is internally multiplied by the velocity scaling factor (SCLV) to convert user units/sec to motor steps/sec.

Servos: If scaling is not enabled (SCALEØ), the velocity value is entered in encoder or resolver revs/sec, LDT inches/sec, or ANI volts/sec; encoder/resolver and LDT values are internally multiplied by the encoder/resolver resolution (ERES) value or the LDT resolution (LDTRES) value to obtain a velocity value in steps/sec for the motion trajectory calculations. If scaling is enabled (SCALE1), the entered velocity value is internally multiplied by the velocity scaling factor (SCLV) to convert user units/sec to encoder, resolver, LDT, or ANI steps/sec.

Example:

```
IF(2V<25)       ; If the programmed velocity on axis 2 is less than 25
                ; units/sec, then do the statements between the IF and NIF
VAR1=2V*2       ; Variable 1 = programmed velocity of axis 2 times 2
V,(VAR1)        ; Set the velocity on axis 2 to the value of variable 1
NIF             ; End the IF statement
```

VAR — Numeric Variable Assignment

Type	Variable
Syntax	<!>VAR<i><=r>
Units	i = variable number, r = number or expression
Range	i = 1 - 100 (AT6n00), 1-200 (AT6n00-M) or 1 - 150 (AT6n50, 610n, 615n, 620n, 625n, & 6270), r = ±999,999,999.99999999
Default	n/a
Response	VAR1: *VAR1=+0.0
See Also	VARB, VARCLR, VARS, WRVAR

Product	Rev
AT6n00	1.1
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.2
625n	1.0
6270	1.0

Numeric variables can be used to store any real number value, with a range from -999,999,999.99999999 to +999,999,999.99999999. The information is assigned to the variable with the equal sign (e.g., VAR1=32.3).

Stand-alone 6000 products: All variables (numeric [VAR], binary [VARB], and string [VARS]) are automatically stored in battery-backed RAM.

Variables are also used in conjunction with mathematical (=, +, -, *, /, SQRT), trigonometric (ATAN, COS, PI, SIN, TAN), and bitwise operators(&, |, ^, ~). For example, VAR1=(3+4-7*4/4+3-2/1.5)*3.

Each variable expression must be less than 80 characters in length, including the VAR1= part of the expression.

Numeric data can also be read into a variable, through the use of the READ , DAT, or TW commands (e.g., VAR1=READ1).

All variables can be used within commands that require a real or integer value. For example, the A command requires real values for acceleration, therefore the command A(VAR1),1Ø,12,(VAR2) is legal. Indirect variable assignments are also legal; (e.g., VAR(VAR1)=5 or VAR(VAR2)=VAR(VAR4)).

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

Example:

```
VAR1=2*PI           ; Set Variable 1 to 2p
D(VAR2),,(VAR3)     ; Set the distance value on axis 1 equal to variable 2,
                    ; and the distance on axis 3 equal to variable 3
```

Indirect Variables: Numeric variables can be used indirectly. Only one level of indirection is possible (e.g., VAR(VAR(VARn)) is not a legal command). The example below shows how indirect variables are used to clear 50 variables (from 1 to 50).

Example:

```
VAR51 = 1           ; Set Variable 51 to 1
REPEAT              ; Begin repeat/until loop
VAR(VAR51) = 0      ; Clear variables (e.g., if VAR51 = 8,
                    ; then VAR(VAR51)=0 is equivalent to VAR8=0)
VAR51 = VAR51 + 1   ; Increment counter
UNTIL (VAR51 = 51)  ; End repeat/until loop
```

VARB — Binary Variable Assignment

Type	Variable
Syntax	<!>VARB<i><=bb...bbb> (32 bits)
Units	i = variable number
Range	i = 1-100 (AT6n00); 1-25 (AT6n50, 610n, 615n, 620n, 625n, & 6270) b = 0, 1, X, or x
Default	n/a
Response	VARB1: *VARB1=XXXX_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX
See Also	VAR, VARCLR, VARS, VCVT, WRVARB

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

Binary variables can be used to store any 32-bit or less binary value. The 32-bit binary value must be in the form of 32 ones, zeros, or Xs. The information is assigned to the binary variable with the equal sign.

Stand-alone 6000 products: All variables (numeric [VAR], binary [VARB], and string [VARS]) are automatically stored in battery-backed RAM.

Example: VARB1=b1111Ø000Ø1111XXXX1111Ø000Øxxxx1111
Notice that the letter b is required. The b signifies binary, 1's, Ø's, and X's only.

Example: VARB1=h7F4356A3
Notice that the letter h is required. The h signifies hexadecimal, Ø-9, A-F only.

Binary variables are also used in conjunction with bitwise operators (&, |, ^, and ~).
Example: VARB1=VARB2 | VARB3 & b1111Ø000Ø11Ø0Ø1

The expression must be less than 80 characters in length, including the (VARB1=b or VARB1=h) part of the expression.

All binary variables can be used to set bits for commands that require at least 4 bits of binary information. For example, the OUT command requires 24 bits of binary information, therefore the command OUT(VARB1) is legal.

Rule of Thumb for command value substitutions: If the command syntax shows that the command field requires a binary value (denoted by <b>), you can use the VARB substitution.

Example:
```
VARB1=b1110 & hA   ; Binary variable 1 is set to binary 1110 bitwise
                   ; "AND"ed with hexadecimal A
VARB1=IN.7         ; Binary variable 1 is set to input bit 7
OUT(VARB1)         ; Assign the value of binary variable 1 to the outputs
```

VARCLR — Variable Clear

Type	Variable
Syntax	<!>VARCLR
Units	n/a
Range	n/a
Default	n/a
Response	n/a
See Also	VAR, VARB, VARS

Product	Rev
AT6n00	3.4
AT6n50	3.4
610n	4.0
615n	3.4
620n	3.4
625n	3.4
6270	3.4

VARCLR resets all numeric variables (VAR), binary variables (VARB), and string variables (VARS) to their factory default values:

Numeric variables are set to 0.0
Binary variables are set to bxxxx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx
String variables are set to ""

VARS — String Variable Assignment

		Product	Rev
Type	Variable		
Syntax	<!>VARS<i><="message">	AT6n00	1.0
Units	i = variable number, message = text string	AT6n50	1.0
Range	i = 1-100 (AT6n00); 1-25 (AT6n50, 610n, 615n, 620n, 625n, & 6270)	610n	4.0
	Message = up to 20 characters	615n	1.0
Default	n/a	620n	1.0
Response	VARS1: *VARS1="Hi John"	625n	1.0
See Also	', [\ ], EOT, [READ], VAR, VARB, VARCLR, VCVT, WRITE, WRVARS	6270	1.0

String variables can be assigned a character string up to 20 characters long. The characters within the string can be any character except the quote ("), the semicolon (;), and the colon (:). The backslash character (\) immediately followed by a number is okay.

Stand-alone 6000 products: All variables (numeric [VAR], binary [VARB], and string [VARS]) are automatically stored in battery-backed RAM.

To place specific control characters that are not directly available on the keyboard within a character string, use the backslash character (\), followed by the control character's ASCII decimal equivalent. Multiple control characters can be sent.

For example, to set the string for variable #1 equal to HI MOM<cr>, use the command VARS1="HI MOM\13" where \13 corresponds to the carriage return character.

Common characters and their ASCII equivalent value:

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

Example:

```
VARS1="Enter velocity >"   ; Assign a message to string variable #1
VAR2=READ1                 ; Transmit string variable 1, and wait for numeric
                           ; data entered in the format of !'<data>.
                           ; Once numeric data is received, place it in
                           ; numeric variable 2.
                           ; Example of data entry is to type "!'10.0", which
                           ; will assign numeric variable 2 the value 10.00
```

VCVT( ) — Variable Type Conversion

		Product	Rev
Type	Operator (Mathematical)		
Syntax	See below	AT6n00	2.1
Units	n/a	AT6n50	1.0
Range	n/a	610n	4.0
Default	n/a	615n	1.0
Response	n/a	620n	2.1
See Also	VAR, VARB	625n	1.1
		6270	1.0

Using the Variable Type Conversion (VCVT) operator, you can convert numeric values to binary values, and vice versa. The operation is a signed operation as the binary value is interpreted as a two's complement number, with the least significant bit (LSB) on the left and the most significant bit (MSB) on the right. A *don't care* (X) in a binary value will be interpreted as a zero (Ø).

If the mathematical statement's result is a numeric value, then VCVT converts binary values to numeric values. If the statement's result is a binary value, then VCVT converts numeric values to binary values.

Numeric-to-Binary Conversion:

```
VAR1=-5           ; Set numeric variable value = -5
VARB1=VCVT(VAR1)  ; Convert the numeric value to a binary value and
                  ; store in VARB1
VARB1             ; Display value of VARB1.  The response should be:
                  ;    *VARB1=1101_1111_1111_1111_1111_1111_1111_1111

VAR1=25           ; Set numeric variable value = 25
VARB1=VCVT(VAR1)  ; Convert the numeric value to a binary value and
                  ; store in VARB1
VARB1             ; Display value of VARB1.  The response should be:
                  ;    *VARB1=1001_1000_0000_0000_0000_0000_0000_0000
```

Binary-to-Numeric Conversion:

```
VARB1=b0010_0110_0000_0000_0000_0000_0000_0000  ; Set binary variable = +100.0
VAR1=VCVT(VARB1)   ; Convert the binary value to a numeric value
VAR1               ; *VAR1=+100.0
```

[VEL] Velocity (Commanded) Assignment

Type	Assignment or Comparison
Syntax	See below
Units	n/a
Range	n/a
Default	n/a
Response	n/a
See Also	SCALE, SCLV, SFB, TVEL, TVELA, V, [V], VELA

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The VEL operator is used to compare the current *commanded* velocity to another value or variable, or to assign the current commanded velocity to a variable. The velocity value used in any comparison, or in any assignment statement is the current commanded velocity value, not the *programmed* velocity (V) or the actual velocity as measured from the feedback device (VELA). .

Syntax: VARn=aVEL where n is the variable number, and a is the axis number, or [VEL] can be used in an expression such as IF(2VEL>4). When assigning the current velocity value to a variable, an axis specifier must always precede the assignment (VEL) command (e.g., VAR1=1VEL). When making a comparison to the current velocity, an axis specifier must also be used, or else it will default to axis 1 (e.g., IF(1VEL<2Ø)).

Steppers: The value is always in reference to motor steps, not encoder steps, regardless of the ENC command setting. If scaling is not enabled (SCALEØ), the velocity value represents motor revs/sec; this value is internally multiplied by the drive resolution (DRES) value to obtain a velocity value in motor steps/sec for the motor trajectory calculations. If scaling is enabled (SCALE1), the velocity value is internally multiplied by the velocity scaling factor (SCLV) to convert user units/sec to motor steps/sec.

Servos: **The velocity value is the current command velocity as calculated by the DSP's move profile routine; it is not the programmed velocity (V).**

If scaling is not enabled (SCALEØ), the velocity value represents encoder revs/sec, LDT inches/sec, or ANI volts/sec; encoder and LDT values are internally multiplied by the encoder resolution (ERES) value or the LDT resolution (LDTRES) value to obtain a velocity value in steps/sec for the motion trajectory calculations. If scaling is enabled (SCALE1), the velocity value is internally multiplied by the velocity scaling factor (SCLV).

Example:

```
IF(2VEL<25)     ; If the current velocity on axis 2 is less than 25 units/sec,
                ; then do the statements between the IF and NIF
  VAR1=2V*2     ; Variable 1 = programmed velocity of axis 2 times 2
NIF             ; End the IF statement
```

[VELA] Velocity (Actual) Assignment

Type	Assignment or Comparison
Syntax	See below
Units	n/a
Range	n/a
Default	n/a
Response	n/a
See Also	ERES, SCLV, TVEL, TVELA, [V], V, [VEL]

Product	Rev
AT6n00	4.1
OEM-AT6n00	n/a
AT6n50	4.1
610n	4.0
615n	4.1
620n	4.1
625n	4.1
6270	4.1

The VELA operator is used to compare the current *actual* velocity (as derived from the feedback device) to another value or variable, or to assign the current velocity to a variable. If the programmed velocity information is required, refer to the [V] operator; if the current commanded velocity information is required, refer to the [VEL] operator.

The sign determines the direction of motion. You can use the VELA operator at all times; therefore, even if no motion is being commanded, TVELA will still report a non-zero value as it detects the servoing action.

Syntax: VARn=aVELA where n is the variable number, and a is the axis number, or [VELA] can be used in an expression such as IF(2VELA>4). When assigning the current velocity value to a variable, an axis specifier must always precede the assignment (VELA) command (e.g., VAR1=1VELA). When making a comparison to the current velocity, an axis specifier must also be used, or else it will default to axis 1 (e.g., IF(1VELA<2Ø)).

Units of Measure:

Steppers: The velocity is always revs/sec (actual velocity in steps/sec multiplied by the ERES value). The VELA command does not reflect the actual velocity during feedrate override (FR).

Servos: If scaling is enabled (SCALE1), the velocity value will be scaled by the velocity scaling factor (SCLV). If scaling is not enabled (SCALEØ), the value returned will be in encoder or resolver revs/sec, LDT inches/sec, or ANI volts/sec.

Example:

```
IF(2VELA<25)   ; If the current velocity on axis 2 is less than 25 units/sec,
               ; then do the statements between IF and NIF
  VAR1=2V*2    ; Variable 1 = programmed velocity of axis 2 times 2
NIF            ; End the IF statement
```

VF Final Velocity

Type	Compiled Motion
Syntax	<!><@>VF<r>,<r>,<r>,<r>
Units	n/a
Range	0 (non-zero values result in error message)
Default	0
Response	n/a
See Also	FOLRN, FOLRD, FOLMAS, FOLMD, GOBUF, SCLD, FOLEN, V

Product	Rev
AT6n00	4.1
AT6n50	4.1
610n	4.0
615n	4.1
620n	4.1
625n	4.1
6270	n/a

The Final Velocity (VF) command designates that the motor will move the load the programmed distance in a preset GOBUF segment, completing the move at a final speed of zero. VF applies only to the next (subsequent) GOBUF, which marks an intermediate "end of move" within a profile. VF is used only in conjunction with the GOBUF command. Normal preset GO moves always finish with zero velocity.

The VF command remains in effect for the affected axis until a GOBUF is executed on that axis, or until you issue a RESET command.

VF will override the Start/Stop Velocity (SSV) for a segment which has VF defined.

Any non-zero value that is entered for VF will result in an immediate error message.

[VMAS] Current Master Velocity

		Product	Rev
Type	Following and Assignment or Comparison	AT6n00	3.0
Syntax	See below	AT6n50	3.0
Units	n/a	610n	4.0
Range	n/a	615n	3.0
Default	n/a	620n	3.0
Response	n/a	625n	3.0
See Also	ENC, FFILT, FMCNEW, FMCP, FOLMAS, FOLMD, SCALE, SCLMAS, TVMAS	6270	3.0

The Master Velocity (VMAS) command is used to assign the master velocity value to a variable, or to make a comparison against another value. **The master must be assigned first (FOLMAS command) before this command will be useful.**

Syntax: VARn=aVMAS where n is the variable number and a is the axis number, or VMAS can be used in an expression such as IF(2VMAS>1Ø). The VMAS command must be used with an axis specifier, or it will default to axis 1 (e.g., VAR1=1VMAS, IF(2VMAS>5), etc.).

The precision of the VMAS value is dependent upon the FFILT filter value.

If scaling is enabled (SCALE1), the velocity value is scaled by the master scaling factor (SCLMAS). If scaling is disabled (SCALEØ), the velocity value is in counts/sec.

Example:

```
IF(2VMAS>4.3)   ; If the master of axis 2 is traveling at more than
                ; 4.3 user units/sec then do the IF statement
  OUT.12=b1     ; Set output #12 to 1
NIF             ; End of IF statement
VAR14=3VMAS     ; Set VAR14 to axis 3's master velocity
```

WAIT( ) Wait for a Specific Condition

		Product	Rev
Type	Program Flow Control	AT6n00	1.0
Syntax	<!>WAIT(expression)	AT6n50	1.0
Units	n/a	610n	4.0
Range	Up to 80 characters (including parentheses)	615n	1.0
Default	n/a	620n	1.0
Response	n/a	625n	1.0
See Also	FMCLEN, FMCNEW, FMCP, GOWHEN, IF, NWHILE, REPEAT, [SS], T, TSS, UNTIL, WHILE	6270	1.0

The Wait for a Specific Condition (WAIT) command is used to wait for a specific expression to evaluate true. No commands, except for immediate commands, after the WAIT command will be processed until the expression contained within the parentheses of the WAIT command evaluates true. The COMEXC command has no effect on the WAIT command.

All logical operators (AND, OR, NOT), and all relational operators (=, >, >=, <, <=, <>) can be used within the WAIT( ) expression. There is no limit on the number of logical operators, or on the number of relational operators allowed within a single WAIT( ) expression.

The limiting factor for the WAIT( ) expression is the command length. The total character count for the WAIT( ) command and expression cannot exceed 80 characters. For example, if you add all the letters in the WAIT command and the letters within the ( ) expression, including the parenthesis and excluding the spaces, this count must be less than or equal to 80.

All assignment operators (A, AD, ANV, AS, CNT, D, ER, IN, INO, LIM, MOV, OUT, PC, PCE, PCM, PE, PER, PM, SS, TIM, US, V, VEL, etc.) can be used within the WAIT( ) expression.

Example:

```
MC1             ; Mode continuous
COMEXC1         ; Enable continuous command mode
GO1             ; Initiate motion on axis 1
WAIT(IN=b1)     ; Wait for input 1 to be active
S1              ; Stop motion on axis 1
WAIT(MOV=b0)    ; Wait for motion complete on axis 1
COMEXC0         ; Disable continuous command execution mode
```

WHILE( ) WHILE Statement

Type	Program Flow Control or Conditional Branching
Syntax	<!>WHILE(expression)
Units	n/a
Range	Up to 80 characters (including parentheses)
Default	n/a
Response	n/a
See Also	IF, JUMP, NWHILE, REPEAT, UNTIL

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The While Statement (WHILE) command, in conjunction with the NWHILE command, provide a means of conditional program flow. The WHILE command marks the beginning of the conditional statement, the NWHILE command marks the end. If the expression contained within the parenthesis of the WHILE command evaluates true, then the commands between the WHILE and NWHILE are executed, and continue to execute as long as the expression evaluates true. If the expression evaluates false, then program execution jumps to the first command after the NWHILE. Up to 16 levels of WHILE ... NWHILE commands may be nested.

Programming order: WHILE(expression) ...commands... NWHILE

NOTE: Be careful about performing a GOTO between WHILE and NWHILE. Branching to a different location within the same program will cause the next WHILE statement encountered to be nested within the previous WHILE statement, unless a NWHILE command has already been encountered. The JUMP command should be used in this situation.

All logical operators (AND, OR, NOT), and all relational operators (=, >, >=, <, <=, <>) can be used within the WHILE( ) expression. There is no limit on the number of logical operators, or on the number of relational operators allowed within a single WHILE( ) expression.

The limiting factor for the WHILE( ) expression is the command length. The total character count for the WHILE( ) command and expression cannot exceed 80 characters. For example, if you add all the letters in the WHILE command and the letters within the ( ) expression, including the parenthesis and excluding the spaces, this count must be less than or equal to 80.

All assignment operators (A, AD, ANV, AS, CNT, D, ER, IN, INO, LIM, MOV, OUT, PC, PCE, PCM, PE, PER, PM, SS, TIM, US, V, VEL, etc.) can be used within the WHILE( ) expression.

Example:

```
WHILE(IN=b1X0)  ; While input 1 = 1, input 3 = Ø, execute commands between
                ; WHILE and NWHILE
T5              ; Wait 5 seconds
TPE             ; Transfer position of all encoders
NWHILE          ; End WHILE statement
WHILE(1ANV<2.3) ; While analog channel 1's voltage is less than 2.3 volts,
                ; execute commands between WHILE and NWHILE
TPM             ; Transfer position of all motors
NWHILE          ; End WHILE statement
```

WRITE — Write a Message

Type	Communication Interface
Syntax	<!>WRITE"<message>"
Units	n/a
Range	Up to 69 characters (may not use ", ; or :)
Default	n/a
Response	WRITE"message": message
See Also	[\ ], EOT, PORT, [READ], VARS, WRVAR, WRVARB, WRVARS

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Write a Message (WRITE) command provides an efficient way of transmitting message strings to the PC-AT bus, or out the RS-232C port. These messages can then be used by the operating program. The EOT command characters will be transmitted after the message.

Each message can be assigned a character string up to 69 characters long. The characters within the string can be any character except the quote ("), the colon (:), and the asterisk (*).

To place specific control characters that are not directly available on the keyboard within the character string, use the backslash character (\), followed by the control character's ASCII decimal equivalent. Multiple control characters can be sent. For example, to set the message equal to HI MOM<cr>, use the command WRITE"HI MOM\13" where \13 corresponds to the carriage return character. Common characters and their ASCII equivalent values are listed below:

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

Example:

```
WRITE"It's a wonderful life!"    ; Send the message "It's a wonderful life!"
```

WRVAR — Write a Numeric Variable

Type	Communication Interface
Syntax	<!>WRVAR<i>
Units	i = variable number
Range	i = 1-100 (AT6n00), 1-200 (AT6n00-M); 1-150 (AT6n50, 610n, 615n, 620n, 625n, & 6270)
Default	n/a
Response	WRVAR1: +0.0
See Also	EOT, [READ], VAR, WRITE, WRVARB, WRVARS

Product	Rev
AT6n00	1.0
AT6n50	1.0
610n	4.0
615n	1.0
620n	1.0
625n	1.0
6270	1.0

The Write a Numeric Variable (WRVAR) command transfers one of the numeric variables (VAR) to the PC-AT bus, or out the RS-232C port, depending on the 6000 Series product. Only the value and the EOT command characters are transmitted.

Example:

```
VAR1=100        ; Set variable 1 equal to 100
WRVAR1          ; Transmit variable 1 (the value +100.0 is transmitted)
```

WRVARB Write a Binary Variable

		Product	Rev
Type	Communication Interface	AT6n00	1.0
Syntax	<!>WRVARB<i>	AT6n50	1.0
Units	i = variable number	610n	4.0
Range	i = 1-100 (AT6n00); 1-25 (AT6n50, 610n, 615n, 620n, 625n, & 6270)	615n	1.0
Default	n/a	620n	1.0
Response	WRVARB1: XXXX_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX	625n	1.0
See Also	EOT, [READ], VARB, WRITE, WRVAR, WRVARS	6270	1.0

The Write a Binary Variable (WRVARB) command transfers one of the binary variables (VARB) to the PC-AT, or out the RS-232C port, depending on the 6000 Series product. Only the binary value and the EOT command characters are transmitted.

Example:

```
VARB1=b1101      ; Set binary variable 1 to 1101
WRVARB1          ; Transmit binary variable 1
                 ; (value transmitted =1101_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX)
```

WRVARS Write a String Variable

		Product	Rev
Type	Communication Interface	AT6n00	1.0
Syntax	<!>WRVARS<i>	AT6n50	1.0
Units	i = variable number	610n	4.0
Range	i = 1-100 (AT6n00); 1-25 (AT6n50, 610n, 615n, 620n, 625n, & 6270)	615n	1.0
Default	n/a	620n	1.0
Response	WRVARS1: No response until a string is placed in VARS1	625n	1.0
See Also	EOT, [READ], VARS, WRITE, WRVAR, WRVARB	6270	1.0

The Write a String Variable (WRVARS) command transfers one of the variable strings (VARS) to the PC-AT, or out the RS-232C port, depending on the 6000 Series product. Only the string and the EOT command characters are transmitted.

Example:

```
VARS1="John L"  ; Set string variable 1 = "John L"
WRVARS1         ; Transmit string variable 1 (string "John L" is transmitted)
```

XONOFF Enable/Disable XON / XOFF

		Product	Rev
Type	Communication Interface	AT6n00	n/a
Syntax	<!>XONOFF<b>	AT6n50	n/a
Units	n/a	610n	4.0
Range	0 (disable), 1 (enable)	615n	4.1
Default	1 for COM1, 0 for COM2 (PORT command setting determines which COM port's XONOFF setting is checked)	620n	4.1
Response	XONOFF *XONOFF1	625n	4.1
See Also	], [, BOT, DRPCHK, E, EOT, ERRBAD, ERROK, PORT	6270	4.1

Use the XONOFF command to enable or disable XON/XOFF (ASCII handshaking).

XONOFF1 enables XON/XOFF, which allows the 6000 product to recognize ASCII handshaking control characters. When XON/XOFF is enabled, ASCII 17 or ^Q is a signal to start sending characters; ASCII 19 or ^S is a signal to stop sending characters. XONOFFØ disables XON/XOFF.

The PORT command determines which COM port is affected by the XONOFF command. Each port will track its XON/XOFF values

RS-485 Multi-drop: If you are using RS-485 multi-drop, disable XON/XOFF by executing the PORT2 command followed by the XONOFFØ command.

NOTE: If COM1 and COM2 are not clearly labeled on your product, COM1 is the RS-232 connector (or Rx, Tx, GND terminals on the AUX connector); COM2 is the RP240 connector.

Appendix A: 6000 Series Command Compatibility

(Revision 4.5)

Command	Description	AT6200	AT6400	AT6250	AT6450	610n	615n	620n	625n	6270
[!]	Immediate Command Identifier	x	x	x	x	x	x	x	x	x
[@]	Global Command Identifier	x	x	x	x	x	x	x	x	x
;	Begin Comment	x	x	x	x	x	x	x	x	x
$	Label Declaration	x	x	x	x	x	x	x	x	x
[#]	Step Through a Program	x	x	x	x	x	x	x	x	x
'	Enter Interactive Data	x	x	x	x	x	x	x	x	x
[.]	Bit Select	x	x	x	x	x	x	x	x	x
["]	Begin and End String	x	x	x	x	x	x	x	x	x
[\ ]	ASCII Character Designator	x	x	x	x	x	x	x	x	x
[=]	Assignment or Equivalence	x	x	x	x	x	x	x	x	x
[>]	Greater Than	x	x	x	x	x	x	x	x	x
[>=]	Greater Than or Equal	x	x	x	x	x	x	x	x	x
[<]	Less Than	x	x	x	x	x	x	x	x	x
[<=]	Less Than or Equal	x	x	x	x	x	x	x	x	x
[<>]	Not Equal	x	x	x	x	x	x	x	x	x
[( )]	Operation Priority Level	x	x	x	x	x	x	x	x	x
[+]	Addition	x	x	x	x	x	x	x	x	x
[-]	Subtraction	x	x	x	x	x	x	x	x	x
[*]	Multiplication	x	x	x	x	x	x	x	x	x
[/]	Division	x	x	x	x	x	x	x	x	x
[&]	Boolean And	x	x	x	x	x	x	x	x	x
[|]	Boolean Inclusive Or	x	x	x	x	x	x	x	x	x
[^]	Boolean Exclusive Or	x	x	x	x	x	x	x	x	x
[~( )]	Boolean Not	x	x	x	x	x	x	x	x	x
[<<]	Shift from Right to Left (bit 32 to bit 1)	x	x	x	x	x	x	x	x	x
[>>]	Shift from Left to Right (bit 1 to bit 32)	x	x	x	x	x	x	x	x	x
[	Send Response to Both COM Ports					x	x	x	x	x
]	Send Response to Alternate COM Port					x	x	x	x	x
A	Acceleration	x	x	x	x	x	x	x	x	x
[A]	Acceleration [operator]	x	x	x	x	x	x	x	x	x
AA	Acceleration, S-curve			x	x		x		x	x
AD	Deceleration	x	x	x	x	x	x	x	x	x
[AD]	Deceleration [operator]	x	x	x	x	x	x	x	x	x
ADA	Deceleration, S-curve			x	x		x		x	x
ADDR	Auto-Address Multiple Serial Units					x	x	x	x	x
[AND]	And [operator]	x	x	x	x	x	x	x	x	x
[ANI]	ANI Analog Input Voltage [operator]			ANI	ANI		ANI		ANI	ANI
ANIPOL	ANI Analog Input Polarity									ANI
[ANV]	Analog Input Channel Voltage [operator]	S	S	x	x			x	x	x
ANVO	Analog Input Channel Voltage Override	S	S	x	x			x	x	x
ANVOEN	Analog Input Channel Override Enable	S	S	x	x			x	x	x
[AS]	Axis Status [operator]	x	x	x	x	x	x	x	x	x
[ASX]	Axis Status, Extended [operator]	x	x	x	x	x	x	x	x	x
[ATAN()]	Arc Tangent [operator]	x	x	x	x	x	x	x	x	x
BOT	Beginning of Transmission Characters	x	x	x	x	x	x	x	x	x
BP	Set a Program Break Point	x	x	x	x	x	x	x	x	x
BREAK	Terminate Program Execution	x	x	x	x	x	x	x	x	x
C	Continue Command Execution	x	x	x	x	x	x	x	x	x
[CA]	Captured ANI Input Voltage [operator]			ANI	ANI		ANI		ANI	ANI
CMDDIR	Commanded Direction Voltage	x	x			x	x	x		x
[CNT]	Counter Value [operator]	S	S			x		x		
CNTE	Hardware Up/Down Counter Input	S	S			x		x		
CNTINT	Counter Value to Interrupt PC-AT	S	S							

S = applicable only to standard (not OEM) version
ANI = applicable only to -ANI version

Command	Description	AT6200	AT6400	AT6250	AT6450	610n	615n	620n	625n	6270
CNTR	Reset Hardware Up/Down Counter	S	S			X		X		
COMEXC	Continuous Command Processing Mode	X	X	X	X	X	X	X	X	X
COMEXK	Continue Execution on Kill	X	X	X	X					
COMEXL	Continue Execution on End-of-Travel Limit	X	X	X	X	X	X	X	X	X
COMEXP	Continue Execution on In Position Input	S	S					X		
COMEXR	Continue Motion on Pause/Continue Input	X	X	X	X	X	X	X	X	X
COMEXS	Continue Execution on Stop	X	X	X	X	X	X	X	X	X
[COS()]	Cosine [operator]	X	X	X	X	X	X	X	X	X
D	Distance	X	X	X	X	X	X	X	X	X
[D]	Distance [operator]	X	X	X	X	X	X	X	X	X
[DAC]	Value of DAC Output Voltage [operator]			X	X		X		X	X
DACLIM	DAC Output Voltage Limit			X	X		X		X	X
DACMIN	Minimum DAC Output Voltage									X
DACTDP	Active Damping					X				
DAREN	Anti-Resonance					X				
[DAT]	Data Assignment [operator]	X	X	X	X	X	X	X	X	X
DATA	Data Statement	X	X	X	X	X	X	X	X	X
[DATP]	Data Program	X	X	X	X	X	X	X	X	X
DATPTR	Set Data Pointer	X	X	X	X	X	X	X	X	X
DATRST	Reset Data Pointer	X	X	X	X	X	X	X	X	X
DATSIZ	Data Program Size	X	X	X	X	X	X	X	X	X
DATTCH	Data Teach	X	X	X	X	X	X	X	X	X
DAUTOS	Automatic Current Standby Mode					X				
DCLEAR	Clear RP240 Display					X	X	X	X	X
DEF	Begin Definition of Program	X	X	X	X	X	X	X	X	X
DEL	Delete Program	X	X	X	X	X	X	X	X	X
DELVIS	Electronic Viscosity					X				
DJOG	Enable RP240 Jog Mode					X	X	X	X	X
DLED	Turn RP240 LEDs ON/OFF					X	X	X	X	X
DMTIND	Motor Inductance					X				
DMTSTT	Motor Static Torque					X				
DPASS	Change RP240 Password					X	X	X	X	X
DPCUR	Position RP240 Display Cursor					X	X	X	X	X
[DPTR]	Data Pointer Location [operator]	X	X	X	X	X	X	X	X	X
[DREAD]	Read RP240 Numeric Data [operator]					X	X	X	X	X
[DREADF]	Read RP240 Function Key [operator]					X	X	X	X	X
DREADI	RP240 Data Read, Immediate Mode					X	X	X	X	X
DRES	Drive Resolution	X	X			X		X		
DRESET	Drive Reset						X			
DRFLVL	Drive Fault Input, Active Level	X	X	X	X			X	X	X
DRIVE	Drive Enable/Disable	S	S	X	X	X	X	X	X	X
DRPCHK	RP240 COM Port Check					X	X	X	X	X
DVAR	Display Numeric Variable on RP240					X	X	X	X	X
DWAVEF	Waveform					X				
DWRITE	Write Text to RP240					X	X	X	X	X
E	Enable Serial Communication					X	X	X	X	X
ECHO	Enable Communication Echo					X	X	X	X	X
ELSE	Else Condition of IF Statement	X	X	X	X	X	X	X	X	X
EMOVDB	Encoder Move Deadband	S	S			X		X		
ENC	Encoder/Motor Step Mode	S	S			X		X		
ENCPOL	Encoder Polarity	S	S			X		X		X
END	End Definition of Program	X	X	X	X	X	X	X	X	X
EOL	End-of-Line Termination Characters	X	X	X	X	X	X	X	X	X
EOT	End-of-Transmission Characters	X	X	X	X	X	X	X	X	X
EPM	Position Maintenance Mode	S	S			X		X		
EPMDB	Position Maintenance Deadband	S	S			X		X		
EPMG	Position Maintenance Gain Factor	S	S			X		X		
EPMV	Position Maintenance Maximum Velocity	S	S			X		X		
[ER]	Error Status [operator]	X	X	X	X	X	X	X	X	X

S = applicable only to standard (not OEM) version
ANI = applicable only to -ANI version

Command	Description	AT6200	AT6400	AT6250	AT6450	610n	615n	620n	625n	6270
ERASE	Erase All Programs	x	x	x	x	x	x	x	x	x
ERES	Encoder Resolution	s	s	x	x	x	x	x	x	x
ERRBAD	Error Prompt Characters	x	x	x	x	x	x	x	x	x
ERRDEF	Program Definition Prompt Characters	x	x	x	x	x	x	x	x	x
ERRLVL	Error Detection Level	x	x	x	x	x	x	x	x	x
ERROK	Good Prompt Characters	x	x	x	x	x	x	x	x	x
ERROR	Enable Error Checking	x	x	x	x	x	x	x	x	x
ERRORP	Assign an Error Program	x	x	x	x	x	x	x	x	x
ESDB	Stall Backlash Deadband	s	s			x		x		
ESK	Kill on Stall	s	s			x		x		
ESTALL	Enable Stall Detection	s	s			x		x		
FASTAT	Fast Status Customization	x	x	x	x					
[FB]	Value of Feedback Device [operator]			x	x		x		x	x
FFILT	Following Filter	x	x	x	x	x	x	x	x	x
FMAXA	Slave Maximum Acceleration	x	x			x		x		
FMAXV	Slave Maximum Velocity	x	x			x		x		
FMCLEN	Master Cycle Length	x	x	x	x	x	x	x	x	x
FMCNEW	Restart Master Cycle Counting	x	x	x	x	x	x	x	x	x
FMCP	Initial Master Cycle Position	x	x	x	x	x	x	x	x	x
FOLEN	Enable Following Mode	x	x	x	x	x	x	x	x	x
FOLK	Following Kill, Limitations			x	x		x		x	x
FOLMAS	Assignment of Master to Slave	x	x	x	x	x	x	x	x	x
FOLMD	Master Distance	x	x	x	x	x	x	x	x	x
FOLRD	Denominator of Slave-to-Master Ratio	x	x	x	x	x	x	x	x	x
FOLDN	Numerator of Slave-to-Master Ratio	x	x	x	x	x	x	x	x	x
FOLRNF	Numerator of Final Slave-to-Master Ratio	x	x	x	x	x	x	x	x	x
FOLSND	Following Step and Direction	s	s	x	x	x	x	x	x	x
FPPEN	Enable Master Position Prediction	x	x	x	x	x	x	x	x	x
FR	Enable Feedrate Override	x	x					x		
FRA	Feedrate Override Acceleration	x	x					x		
FRH	Feedrate Override High Channel	s	s					x		
FRL	Feedrate Override Low Channel	s	s					x		
FRPER	Feedrate Override Percentage	x	x					x		
[FS]	Following Status [operator]	x	x	x	x	x	x	x	x	x
FSHFC	Continuous Shift	x	x	x	x	x	x	x	x	x
FSHFD	Preset Shift	x	x	x	x	x	x	x	x	x
GO	Initiate Motion	x	x	x	x	x	x	x	x	x
GOBUF	Store a Compiled Motion Segment	x	x	x	x	x	x	x	x	x
GOL	Initiate Linear Interpolated Motion	x	x	x	x			x	x	x
GOSUB	Call a Subroutine	x	x	x	x	x	x	x	x	x
GOTO	Goto a Program or Label	x	x	x	x	x	x	x	x	x
GOWHEN	Conditional Go	x	x	x	x	x	x	x	x	x
HALT	Terminate Program Execution	x	x	x	x	x	x	x	x	x
HELP	Technical Support Phone Numbers	x	x	x	x	x	x	x	x	x
HOM	Initiate Homing Operation	x	x	x	x	x	x	x	x	x
HOMA	Homing Acceleration	x	x	x	x	x	x	x	x	x
HOMAA	Homing Acceleration, S-curve			x	x		x		x	x
HOMAD	Homing Deceleration	x	x	x	x	x	x	x	x	x
HOMADA	Homing Deceleration, S-curve			x	x		x		x	x
HOMBAC	Backup to Home	x	x	x	x	x	x	x	x	x
HOMDF	Homing Final Direction	x	x	x	x	x	x	x	x	x
HOMEDG	Home Reference Edge	x	x	x	x	x	x	x	x	x
HOMLVL	Home Input, Active Level	x	x	x	x	x	x	x	x	x
HOMV	Homing Velocity	x	x	x	x	x	x	x	x	x
HOMVF	Homing Velocity, Final Approach	x	x	x	x	x	x	x	x	x
HOMZ	Home to Encoder Z Channel	s	s	x	x	x	x	x	x	x
IF()	IF Statement	x	x	x	x	x	x	x	x	x
[IN]	Programmable Input Status [operator]	x	x	x	x	x	x	x	x	x
INDAX	Participating Axes	x	x	x	x			x	x	x

S = applicable only to standard (not OEM) version
ANI = applicable only to -ANI version

Command	Description	AT6200	AT6400	AT6250	AT6450	610n	615n	620n	625n	6270
INDEB	Programmable Input Debounce	x	x	x	x	x	x	x	x	x
INDUSE	Enable User Status	x	x	x	x	x	x	x	x	x
INDUST	User Status Definition	x	x	x	x	x	x	x	x	x
INEN	Enable Specific Programmable Inputs	x	x	x	x	x	x	x	x	x
INFEN	Enable Programmable Input Functions	x	x	x	x	x	x	x	x	x
INFNC	Programmable Input Function Assignment	x	x	x	x	x	x	x	x	x
INLVL	Programmable Input Active Level	x	x	x	x	x	x	x	x	x
[INO]	Other Inputs Status [operator]	s	s	x	x	x	x	x	x	x
INPLC	Establish PLC Data Inputs	s	s	x	x	x	x	x	x	x
INSELP	Enable Program Selection via Inputs	x	x	x	x	x	x	x	x	x
INSTW	Establish Thumbwheel Data Inputs	x	x	x	x	x	x	x	x	x
INTCLR	Clear Interrupt Condition Status	x	x	x	x					
INTHW	Enable Hardware Interrupt	x	x	x	x					
INTSW	Force Software Interrupt	x	x	x	x					
JOG	Enable Jog Mode	x	x	x	x	x	x	x	x	x
JOGA	Jog Acceleration	x	x	x	x	x	x	x	x	x
JOGAA	Jog Acceleration, S-curve			x	x		x		x	x
JOGAD	Jog Deceleration	x	x	x	x	x	x	x	x	x
JOGADA	Jog Deceleration, S-curve			x	x		x		x	x
JOGVH	Jog Velocity, High	x	x	x	x	x	x	x	x	x
JOGVL	Jog Velocity, Low	x	x	x	x	x	x	x	x	x
JOY	Enable Joystick Mode	s	s	x	x			x	x	x
JOYA	Joystick Acceleration	s	s	x	x			x	x	x
JOYAA	Joystick Acceleration, S-curve			x	x				x	x
JOYAD	Joystick Deceleration	s	s	x	x			x	x	x
JOYADA	Joystick Deceleration, S-curve			x	x				x	x
JOYAXH	Joystick Analog Channel, High	s	s	x	x			x	x	x
JOYAXL	Joystick Analog Channel, Low	s	s	x	x			x	x	x
JOYCDB	Joystick Center Deadband	s	s	x	x			x	x	x
JOYCTR	Joystick Center	s	s	x	x			x	x	x
JOYEDB	Joystick End Deadband	s	s	x	x			x	x	x
JOYVH	Joystick Velocity, High	s	s	x	x			x	x	x
JOYVL	Joystick Velocity, Low	s	s	x	x			x	x	x
JOYZ	Joystick Zero (Center)	s	s	x	x			x	x	x
JUMP	Jump to Program or Label (No Return)	x	x	x	x	x	x	x	x	x
K	Kill Motion	x	x	x	x	x	x	x	x	x
<ctrl>K	Kill Motion	x	x	x	x	x	x	x	x	x
KDRIVE	Disable Drive on Kill			x	x		x		x	x
L	Loop	x	x	x	x	x	x	x	x	x
[LDT]	Position of LDT [operator]									x
LDTGRD	LDT Gradient									x
LDTPOL	LDT Polarity									x
LDTRES	LDT Resolution									x
LDTUPD	LDT Update Rate									x
LH	Enable Hardware End-of-Travel Limits	x	x	x	x	x	x	x	x	x
LHAD	Hardware EOT Limits Deceleration	x	x	x	x	x	x	x	x	x
LHADA	Hardware EOT Limits Decel, S-curve			x	x		x		x	x
LHLVL	Hardware EOT Limit Inputs, Active Level	x	x	x	x	x	x	x	x	x
[LIM]	Hardware EOT Limit Inputs, Status	x	x	x	x	x	x	x	x	x
LN	End of Loop	x	x	x	x	x	x	x	x	x
LS	Enable Software End-of-Travel Limits	x	x	x	x	x	x	x	x	x
LSAD	Software EOT Limits, Deceleration	x	x	x	x	x	x	x	x	x
LSADA	Software EOT Limits Decel, S-curve			x	x		x		x	x
LSNEG	Negative-Direction Software EOT Limit	x	x	x	x	x	x	x	x	x
LSPOS	Positive-Direction Software EOT Limit	x	x	x	x	x	x	x	x	x
LX	Terminate Loop	x	x	x	x	x	x	x	x	x
MA	Enable Absolute/Incremental Positioning	x	x	x	x	x	x	x	x	x
MC	Enable Continuous/Preset Positioning	x	x	x	x	x	x	x	x	x
MEMORY	Partition Product Memory	x	x	x	x	x	x	x	x	x

S = applicable only to standard (not OEM) version
ANI = applicable only to -ANI version

Command	Description	AT6200	AT6400	AT6250	AT6450	610n	615n	620n	625n	6270
[MOV]	Axis Moving Status [operator]	x	x	x	x	x	x	x	x	x
NIF	End IF Statement	x	x	x	x	x	x	x	x	x
[NMCY]	Master Cycle Number Status [operator]	x	x	x	x	x	x	x	x	x
[NOT]	Not [operator]	x	x	x	x	x	x	x	x	x
NWHILE	End of WHILE Statement	x	x	x	x	x	x	x	x	x
ONCOND	Enable Program Interrupt (“On”) Conditions	x	x	x	x	x	x	x	x	x
ONIN	On an Input Condition GOSUB	x	x	x	x	x	x	x	x	x
ONP	On Condition Program Assignment	x	x	x	x	x	x	x	x	x
ONUS	On a User Status Condition GOSUB	x	x	x	x	x	x	x	x	x
ONVARA	On Numeric Variable 1 Condition GOSUB	x	x	x	x	x	x	x	x	x
ONVARB	On Numeric Variable 2 Condition GOSUB	x	x	x	x	x	x	x	x	x
[OR]	Or [operator]	x	x	x	x	x	x	x	x	x
OUT	Activate Programmable Outputs	x	x	x	x	x	x	x	x	x
[OUT]	Programmable Outputs Status [operator]	x	x	x	x	x	x	x	x	x
OUTALL	Activate Programmable Outputs, Range	x	x	x	x	x	x	x	x	x
OUTANA	ANA Analog Output Voltage			x	x					
OUTEN	Disable Programmable Outputs	x	x	x	x	x	x	x	x	x
OUTFEN	Enable Programmable Output Functions	x	x	x	x	x	x	x	x	x
OUTFNC	Programmable Output Function Assignment	x	x	x	x	x	x	x	x	x
OUTLVL	Programmable Output Active Level	x	x	x	x	x	x	x	x	x
OUTPA	Output on Position — Axis 1			x	x		x		x	x
OUTPB	Output on Position — Axis 2			x	x				x	x
OUTPC	Output on Position — Axis 3				x					
OUTPD	Output on Position — Axis 4				x					
OUTPLC	Establish PLC Strobe Outputs	x	x	x	x	x	x	x	x	x
OUTTW	Establish Thumbwheel Strobe Outputs	x	x	x	x	x	x	x	x	x
PA	Path Acceleration	x	x	x	x			x	x	x
PAA	Path Acceleration, S-curve			x	x				x	x
PAB	Path Absolute	x	x	x	x			x	x	x
PAD	Path Deceleration	x	x	x	x			x	x	x
PADA	Path Deceleration, S-curve			x	x				x	x
[PANI]	Position of ANI Inputs			ANI	ANI		ANI		ANI	ANI
PARCM	Radius-Specified CCW Arc Segment	x	x	x	x			x	x	x
PARCOM	Origin-Specified CCW Arc Segment	x	x	x	x			x	x	x
PARCOP	Origin-Specified CW Arc Segment	x	x	x	x			x	x	x
PARCP	Radius-Specified CW Arc Segment	x	x	x	x			x	x	x
PAXES	Participating Axes for Contouring	x	x	x	x			x	x	x
[PC]	Position Commanded [operator]			x	x		x		x	x
[PCA]	Position of Captured ANI Input [operator]			ANI	ANI		ANI		ANI	ANI
[PCC]	Captured Commanded Position [operator]			x	x		x		x	x
[PCE]	Position of Captured Encoder [operator]	S	S	x	x	x	x	x	x	x
[PCL]	Position of Captured LDT [operator]									x
[PCM]	Position of Captured Motor [operator]	x	x			x		x		
PCOMP	Compile a Profile	x	x	x	x	x	x	x	x	x
[PE]	Position of Encoder [operator]	S	S	x	x	x	x	x	x	x
[PER]	Position Error [operator]	S	S	x	x	x	x	x	x	x
[PI]	Pi (π) [operator]	x	x	x	x	x	x	x	x	x
PL	Select Path Local/Work Coordinate System	x	x	x	x			x	x	x
PLC	Define Path Local Coordinates	x	x	x	x			x	x	x
PLIN	Move in a Line (Line Segment)	x	x	x	x			x	x	x
PLN	End of Loop, Compiled Motion	x	x	x	x	x	x	x	x	x
PLOOP	Start of Loop, Compiled Motion	x	x	x	x	x	x	x	x	x
[PM]	Position of Motor (Commanded) [operator]	x	x			x		x		
[PMAS]	Position of Master (current cycle) [operator]	x	x	x	x	x	x	x	x	x
PORT	Designate Destination COM Port					x	x	x	x	x
POUT	Compiled Output (Contouring)	x	x	x	x			x	x	x
POUTA	Compiled Output (Compiled Motion), Axis 1	x	x	x	x	x	x	x	x	x
POUTB	Compiled Output (Compiled Motion), Axis 2	x	x	x	x			x	x	x
POUTC	Compiled Output (Compiled Motion), Axis 3		x		x					

S = applicable only to standard (not OEM) version
ANI = applicable only to -ANI version

Command	Description	AT6200	AT6400	AT6250	AT6450	610n	615n	620n	625n	6270
POUTD	Compiled Output (Compiled Motion), Axis 4		x		x					
PPRO	Path Proportional Axis		x		x					
PRTOL	Path Radius Tolerance	x	x	x	x			x	x	x
PRUN	Run a Compiled Profile	x	x	x	x	x	x	x	x	x
PS	Pause Program Execution	x	x	x	x	x	x	x	x	x
PSCLA	Path Acceleration Scale Factor	x	x	x	x			x	x	x
PSCLD	Path Distance Scale Factor	x	x	x	x			x	x	x
PSCLV	Path Velocity Scale Factor	x	x	x	x			x	x	x
PSET	Establish Absolute Position Reference	x	x	x	x	x	x	x	x	x
[PSHF]	Net Position Shift Status [operator]	x	x	x	x	x	x	x	x	x
[PSLV]	Commanded Slave Position [operator]	x	x	x	x	x	x	x	x	x
PTAN	Path Tangent Axis Resolution		x		x					
PUCOMP	Un-Compile a Compiled Profile	x	x	x	x	x	x	x	x	x
PULSE	Step Output Pulse Width	x	x					x		
PV	Path Velocity	x	x	x	x			x	x	x
PWC	Path Work Coordinates	x	x	x	x			x	x	x
RADIAN	Specify Units in Radians or Degrees	x	x	x	x	x	x	x	x	x
RE	Enable Registration	x	x	x	x	x	x	x	x	x
[READ]	Read a Value	x	x	x	x	x	x	x	x	x
REG	Registration Distance	x	x	x	x	x	x	x	x	x
REGLOD	Registration Lockout Distance	x	x	x	x	x	x	x	x	x
REGSS	Registration Single-Shot	x	x	x	x	x	x	x	x	x
REPEAT	REPEAT Statement	x	x	x	x	x	x	x	x	x
RESET	Reset the 6000 Controller	x	x	x	x	x	x	x	x	x
RUN	Begin Executing a Program	x	x	x	x	x	x	x	x	x
S	Stop Motion	x	x	x	x	x	x	x	x	x
SCALE	Enable Scaling Factors	x	x	x	x	x	x	x	x	x
SCLA	Acceleration Scale Factor	x	x	x	x	x	x	x	x	x
SCLD	Distance Scale Factor	x	x	x	x	x	x	x	x	x
SCLMAS	Master Axis Scale Factor	x	x	x	x	x	x	x	x	x
SCLV	Velocity Scale Factor	x	x	x	x	x	x	x	x	x
SD	Streaming Data	x	x							
SDTAMP	Dither Amplitude			x	x		x		x	x
SDTFR	Dither Frequency Ratio			x	x		x		x	x
[SEG]	Number of Free Segment Buffers [operator]	x	x	x	x	x	x	x	x	x
SFB	Select Servo Feedback Source			x	x		x		x	x
SGAF	Gain – Acceleration Feedforward			x	x		x		x	x
SGAFN	Gain – Acceleration Feedforward, Negative									x
SGENB	Enable a Servo Gain Set			x	x		x		x	x
SGI	Gain – Integral Feedback			x	x		x		x	x
SGILIM	Gain – Integral Windup Limit			x	x		x		x	x
SGIN	Gain – Integral Feedback, Negative									x
SGP	Gain – Proportional Feedback			x	x		x		x	x
SGPN	Gain – Proportional Feedback, Negative									x
SGSET	Save a Servo Gain Set			x	x		x		x	x
SGV	Gain – Velocity Feedback			x	x		x		x	x
SGVF	Gain – Velocity Feedforward			x	x		x		x	x
SGVFN	Gain – Velocity Feedforward, Negative									x
SGVN	Gain – Velocity Feedback, Negative									x
[SIN()]	Sine [operator]	x	x	x	x	x	x	x	x	x
SMPER	Maximum Allowable Position Error			x	x		x		x	x
SOFFS	Servo Control Signal Offset			x	x		x		x	x
SOFFSN	Servo Control Signal Offset, Negative									x
[SQRT]	Square Root [operator]	x	x	x	x	x	x	x	x	x
[SS]	System Status [operator]	x	x	x	x	x	x	x	x	x
SSFR	Servo Sampling Frequency Ratio			x	x		x		x	x
SSV	Start/Stop Velocity	x	x			x		x		
SSWD	Setpoint Window Distance									x
SSWG	Setpoint Window Gain Set									x

S = applicable only to standard (not OEM) version
ANI = applicable only to -ANI version

Command	Description	AT6200	AT6400	AT6250	AT6450	610n	615n	620n	625n	6270
STARTP	Start-up Program					X	X	X	X	X
STD	Streaming Interval	X	X							
STEP	Enable Single Step Mode	X	X	X	X	X	X	X	X	X
STREAM	Enable Streaming Mode	X	X							
STRGTD	Target Zone Distance	S	S	X	X	X	X	X	X	X
STRGTE	Enable Target Zone Mode	S	S	X	X	X	X	X	X	X
STRGTT	Target Zone Timeout Period	S	S	X	X	X	X	X	X	X
STRGTV	Target Zone Velocity	S	S	X	X	X	X	X	X	X
T	Time Delay	X	X	X	X	X	X	X	X	X
[TAN()]	Tangent [operator]	X	X	X	X	X	X	X	X	X
TANI	Transfer ANI Analog Input Voltage			ANI	ANI		ANI		ANI	ANI
TANV	Transfer Analog Input Channel Voltage	S	S	X	X			X	X	X
TAS	Transfer Axis Status	X	X	X	X	X	X	X	X	X
TASF	Transfer Axis Status (full-text report)	X	X	X	X	X	X	X	X	X
TASX	Transfer Axis Status, Extended	X	X	X	X	X	X	X	X	X
TASXF	Transfer Axis Status, Extended (full-text)	X	X	X	X	X	X	X	X	X
TCA	Transfer Value of Captured ANI Input			ANI	ANI		ANI		ANI	ANI
TCMDER	Transfer Command Error	X	X	X	X	X	X	X	X	X
TCNT	Transfer Hardware Counter Value	S	S			X		X		
TDAC	Transfer DAC Voltage			X	X		X		X	X
TDIR	Transfer Program Directory	X	X	X	X	X	X	X	X	X
TDPTR	Transfer Data Pointer Status	X	X	X	X	X	X	X	X	X
TER	Transfer Error Status	X	X	X	X	X	X	X	X	X
TERF	Transfer Error Status (full-text report)	X	X	X	X	X	X	X	X	X
TEST	Test Motion	X	X			X		X		
TEX	Transfer Program Execution Status	X	X	X	X	X	X	X	X	X
TFB	Transfer Position of Feedback Devices			X	X		X		X	X
TFS	Transfer Following Status	X	X	X	X	X	X	X	X	X
TFSF	Transfer Following Status (full-test report)	X	X	X	X	X	X	X	X	X
TGAIN	Transfer Servo Gains			X	X		X		X	X
[TIM]	Current Timer Value [operator]	X	X	X	X	X	X	X	X	X
TIMINT	Timer Value to Interrupt PC-AT	X	X	X	X					
TIMST	Start Timer	X	X	X	X	X	X	X	X	X
TIMSTP	Stop Timer	X	X	X	X	X	X	X	X	X
TIN	Transfer Programmable Input Status	X	X	X	X	X	X	X	X	X
TINO	Transfer Other Input Status	S	S	X	X	X	X	X	X	X
TINOF	Transfer Other Input Status (full-text report)	S	S	X	X	X	X	X	X	X
TINT	Transfer Interrupt Status	X	X	X	X					
TLABEL	Transfer Labels	X	X	X	X	X	X	X	X	X
TLDT	Transfer Position of LDT									X
TLIM	Transfer Hardware Travel Limit Status	X	X	X	X	X	X	X	X	X
TMEM	Transfer Memory Usage	X	X	X	X	X	X	X	X	X
TNMCY	Transfer Master Cycle Number	X	X	X	X	X	X	X	X	X
TOUT	Transfer Programmable Output Status	X	X	X	X	X	X	X	X	X
TPANI	Transfer Position of ANI Inputs			ANI	ANI		ANI		ANI	ANI
TPC	Transfer Commanded Position			X	X		X		X	X
TPCA	Transfer Position of Captured ANI Input			ANI	ANI		ANI		ANI	ANI
TPCC	Transfer Captured Commanded Position			X	X		X		X	X
TPCE	Transfer Position of Captured Encoder	S	S	X	X	X	X	X	X	X
TPCL	Transfer Position of Captured LDT									X
TPCM	Transfer Position of Captured Motor	X	X			X		X		
TPE	Transfer Position of Encoder	S	S	X	X	X	X	X	X	X
TPER	Transfer Position Error	S	S	X	X	X	X	X	X	X
TPM	Transfer Position of Motor	X	X			X		X		
TPMAS	Transfer Position of Master (current cycle)	X	X	X	X	X	X	X	X	X
TPROG	Transfer Program Contents	X	X	X	X	X	X	X	X	X
TPSHF	Transfer Net Position Shift	X	X	X	X	X	X	X	X	X
TPSLV	Transfer Commanded Position of Slave	X	X	X	X	X	X	X	X	X
TRACE	Enable Program Trace Mode	X	X	X	X	X	X	X	X	X

S = applicable only to standard (not OEM) version
ANI = applicable only to -ANI version

Command	Description	AT6200	AT6400	AT6250	AT6450	610n	615n	620n	625n	6270
TRANS	Enable Translation Mode	x	x	x	x	x	x	x	x	x
TREV	Transfer Revision Level	x	x	x	x	x	x	x	x	x
TRGFN	Trigger Functions	x	x	x	x	x	x	x	x	x
TSEG	Transfer Number of Free Segment Buffers	x	x	x	x	x	x	x	x	x
TSGSET	Transfer Servo Gain Set			x	x		x		x	x
TSS	Transfer System Status	x	x	x	x	x	x	x	x	x
TSSF	Transfer System Status (full-text report)	x	x	x	x	x	x	x	x	x
TSTAT	Transfer Controller Statistics	x	x	x	x	x	x	x	x	x
TSTLT	Transfer Settling Time	x	x	x	x	x	x	x	x	x
TTIM	Transfer Timer Value	x	x	x	x	x	x	x	x	x
TUS	Transfer User Status	x	x	x	x	x	x	x	x	x
TVEL	Transfer Current Commanded Velocity	x	x	x	x	x	x	x	x	x
TVELA	Transfer Current Actual Velocity	s	s	x	x	x	x	x	x	x
TVMAS	Transfer Current Master Velocity	x	x	x	x	x	x	x	x	x
[TW]	Thumbwheel Assignment [operator]	x	x	x	x	x	x	x	x	x
UNTIL()	Until Part of REPEAT Statement	x	x	x	x	x	x	x	x	x
[US]	User Status [operator]	x	x	x	x	x	x	x	x	x
V	Velocity	x	x	x	x	x	x	x	x	x
[V]	Velocity [operator]	x	x	x	x	x	x	x	x	x
VAR	Numeric Variable Assignment	x	x	x	x	x	x	x	x	x
VARB	Binary Variable Assignment	x	x	x	x	x	x	x	x	x
VARCLR	Clear All Variables	x	x	x	x	x	x	x	x	x
VARS	String Variable Assignment	x	x	x	x	x	x	x	x	x
VCVT()	Variable Type Conversion	x	x	x	x	x	x	x	x	x
[VEL]	Commanded Velocity Assignment [operator]	x	x	x	x	x	x	x	x	x
[VELA]	Actual Velocity Assignment [operator]	s	s	x	x	x	x	x	x	x
VF	Final Velocity	x	x	x	x	x	x	x	x	x
[VMAS]	Velocity of Master [operator]	x	x	x	x	x	x	x	x	x
WAIT()	Wait for a Specific Condition	x	x	x	x	x	x	x	x	x
WHILE()	WHILE Statement	x	x	x	x	x	x	x	x	x
WRITE	Write a Message	x	x	x	x	x	x	x	x	x
WRVAR	Write a Numeric Variable	x	x	x	x	x	x	x	x	x
WRVARB	Write a Binary Variable	x	x	x	x	x	x	x	x	x
WRVARS	Write a String Variable	x	x	x	x	x	x	x	x	x
XONOFF	Enable XON/XOFF ASCII Handshaking					x	x	x	x	x

S = applicable only to standard (not OEM) version
ANI = applicable only to -ANI version

Appendix B: X Series vs. 6000 Series Compatibility

X Series Command List		6000 Series Command List
"	(Quote)	WRITE
#	(Step Sequence)	[#]
;	(Comment Field)	;
A	(Acceleration)	A
AB	(Report Analog Voltage, Binary)	TANV, TANI
AD	(Deceleration)	AD, ADA
AV	(Report Analog Voltage, ASCII)	TANV, TANI
B	(Buffer Status)	TMEM
BCPE	(Buffered Configure Position Error)	ESDB
BCPG	(Buffered Configure Proportional Gain)	EPMG
BCPM	(Buffered Configure Proportional Max)	EPMG
BL	(Backlash)	
BS	(Buffer Status Report)	TMEM
C	(Continue)	C
CG	(Correction Gain)	EPMG
CM	(Set Correction Mode for Position Maintenance)	
CPE	(Configure Position Error)	ESDB
CPG	(Configure Proportional Gain)	EPMG
CPM	(Configure Proportional Max)	EPMG
CR	(Carriage Return)	WRITE"\13"
D	(Distance)	D
DCLR	(Clear Display [RP240])	DCLEAR
DCNT	(Enable/Disable Pause and Continue [RP240])	
DFS	(Display Flags for Servo Parameters)	TAS
DFX	(Display Flags for Indexer Status)	TAS, TSS
DIN	(Disable Inputs)	INEN
DLED	(Turn RP240 LEDs On/Off	DLED
DOUT	(Disable Outputs)	OUTEN
DP	(Distance Point)	
DPA	(Display Position Actual)	TFB, TLDT, TPANI, TPE, TPM
DPC	(Position Cursor [RP240])	DPCUR
DPE	(Display Position Error)	TPER
DR	(Display Parameters)	TSTAT
DRD	(Read Distance Via Parallel I/O)	D, [TW]
DSTP	(Enable/Disable Stop [RP240])	
DTXT	(Display Text on RP240 LCD)	DWRITE
DVA	(Display Actual Velocity)	TVEL, TVELA
DVO	(Display Variable Data on RP240 LCD)	DVAR
DVS	(Display Velocity Setpoint)	
DW	(Deadband Window)	EPMDB
E	(Enable Communications Interface)	E
ELSE	(Else Portion of IF Command)	ELSE
ER	(Encoder Resolution)	ERES
F	(Disable Communication Interface)	E
FAC	(Set Following Synchronization Rate)	
FBS	(Following Base)	

X Series Command List		6000 Series Command List
FC	(Following Learn Count)	
FEN	(Set Following Synchronization Count)	
FIN	(Following Increment)	
FOL	(Following Percent)	
FOR	(Following Ratio)	
FP	(Following Encoder Point)	
FPA	(Following Encoder Absolute Point)	
FR	(Encoder Functions Report)	Every encoder command will report back.
FRD	(Read Following Via Parallel I/O)	
FS	(Encoder Functions Report)	Every encoder command will report back.
FSA	(Set Incremental/Absolute Mode)	MA
FSB	(Set Indexer to Motor/Encoder Mode)	ENC
FSC	(Position Maintenance)	EPM
FSD	(Stop on Stall)	ESK, ESTALL
FSE	(Enable Output #6 on Stall)	ESTALL, OUTFNC
FSF	(Enable Stop on Trigger #6)	INFNC
FSI	(Enable/Disable Following Mode)	
FSK	(Set Following Learn Mode)	
FSL	(Enable/Disable Self Correction Mode)	
FSM	(Set Absolute Encoder)	
FSN	(Set Pulse Following)	
FSP	(Set Tracking Mode)	
[FUN]	(Enable/Read RP240 Function Keys)	[DREADF]
G	(Go)	GO
Gnnn	(Synchronized Go)	GO
GA	(Go Home Acceleration)	HOMA
GD	(Go Defined)	GO, DEF
GDEF	(Configure Move Definition)	GO, DEF
GH	(Go Home)	HOM
GHA	(Go Home Acceleration)	HOMA
GHAD	(Go Home Deceleration)	HOMAD
GHF	(Go Home Final Velocity)	HOMVF
GHV	(Go Home Velocity)	HOMV
GOSUB	(Gosub to a Subroutine)	GOSUB
GOTO	(Go to a Subroutine)	GOTO
^H	(Backspace)	<bksp>
H	(Set Direction)	D
HALT	(Halt)	HALT
I	(Load Move Data)	
ID	(Immediate Distance)	
IF	(IF Command)	IF( )
IN	(Set Input Functions)	INFNC
INL	(Set Input Active Level)	INLVL
INR	(Enable/Disable Registration Input)	INFNC
IO	(Immediate Output)	!OUT
IS	(Input Status)	TIN
IV	(Immediate Velocity)	!V
J	(Enable/Disable Joystick)	JOY
JA	(Jog Acceleration)	JOGA
JAD	(Jog Deceleration)	JOGAD

X Series Command List		6000 Series Command List
JB	(Set Joystick Backlash)	JOYEDB
JD	(Set Joystick Dead band)	JOYCDB
JV	(Set Joystick Backlash Compensation Velocity)	
JVH	(Jog Velocity High)	JOGVH
JVL	(Jog Velocity Low)	JOGVL
JZ	(Set Joystick to Zero)	JOYZ
K	(Kill)	K, <ctrl>K
L	(Loop)	L
LA	(Limit Acceleration)	LHAD
LAD	(Limit Deceleration)	LHAD
LD	(Limit Disable)	LH
LF	(Line Feed)	WRITE"\1Ø"
LRD	(Read Loop Count Via Parallel I/O)	L, [TW]
MA	(Mode Alternate)	D,[~( )],GO
MC	(Mode Continuous)	MC
MN	(Mode Normal)	MC
MPA	(Mode Position Absolute)	MA
MPI	(Mode Position Incremental)	MA
MPP	(Mode Profile Position)	
MR	(Select Motor Resolution)	DRES
MSL	(Identify Clock Source for Timed Data Streaming)	
MSS	(Start Master Clock for Timed Data Streaming)	
MV	(Set Maximum Correction Velocity)	EPMV
N	(End of Loop)	LN
NG	(End Position Profile)	
NIF	(End IF Command)	NIF
[NUM]	(Enable/Read RP240 Numeric Keypad)	[DREAD]
NWHILE	(End WHILE Command)	NWHILE
O	(Output)	OUT
OFF	(Shutdown Drive)	DRIVE
ON	(Activate Drive)	DRIVE
OR	(Report Function Setups)	All homing functions will report settings.
OS	(Report Function Setups)	All homing functions will report settings.
OSA	(Set Encoder Direction)	
OSB	(Backup to Home)	HOMBAC
OSC	(Define Active State of Home Switch)	HOMLVL
OSD	(Define Active State of Encoder's Z Channel Input)	
OSE	(Enable Stall Detect)	ESTALL
OSF	(Set Maximum Joystick Velocity)	JOYVL and JOYVH
OSG	(Set Final Go Home Direction)	HOMDF
OSH	(Reference Edge of Home Switch)	HOMEDG
OUT	(Set Output Functions)	OUTFNC
OUTL	(Output Active Level)	OUTLVL
OUTP	(Output on Position)	WAIT( ), [PM], OUT
P	(Report Incremental Position, ASCII)	
PB	(Report Incremental Position, Binary)	
PF	(Follower Position Report)	
PFZ	(Set Follower Counter to Zero)	
PR	(Report Absolute Position)	TFB, TLDT, TPANI, TPE, TPM
PS	(Pause)	PS

X Series Command List		6000 Series Command List
PX	(Report Encoder Absolute Position, ASCII)	TPE
PXB	(Report Encoder Absolute Position, Binary)	TPE
PZ	(Position Zero)	PSET
Q	(Complete Current Command & Clear Command Buffer)	
QØ	(Exit Streaming Mode)	!STREAM, SD
Q1	(Enter Immediate Velocity Streaming Mode)	STREAM
Q2	(Enter Time-Distance Streaming Mode)	STREAM
Q3	(Enter Time-Velocity Streaming Mode)	
QI	(Report Status of QS Commands)	
QIB	(Interrupt Status Report, Binary)	TINT
QR	(Report QS Command Function Enable Status)	INTHW
QS	(Interrupt on Signal Commands)	INTHW
QSA	(Interrupt on Trigger #1 High)	INTHW and INFNC
QSB	(Interrupt on Move Complete)	INTHW
QSD	(Interrupt Signal on Limit Encountered)	INTHW
QSE	(Interrupt on Ready to Respond)	
QSG	(Interrupt on Command Buffer Full)	INTHW
QSH	(Interrupt on Motor Stall)	INTHW
R	(Request Indexer Status)	TSTAT
RA	(Limit Switch Status Report)	TLIM
RB	(Report Loop, Pause, Shutdown, Trigger Status)	TSS
RC	(Report Closed Loop and Go Home Status)	TAS
REG	(Configure Registration Move)	REG
REPEAT	(Repeat Command)	REPEAT
RG	(Go Home Status Report)	TAS
RIFS	(Return Indexer to Factory Settings)	RESET
RM	(Rate Multiplier in Immediate Velocity Streaming Mode)	
RS	(Report Status of Sequence Execution)	TSS
RSE	(Report Servo Errors)	TAS, TSS
RSIN	(Set Variables Interactively)	VAR, [READ]
RV	(Report Software Part Number)	TREV
S	(Stop)	S
SD	(Streaming Data)	SD
SFL	(Set User Flag)	INTSW
SL	(Software Limits)	LSPOS, LSNEG
SLD	(Software Limit Disable)	LS
SN	(Scan Delay Time)	INSELP
SP	(Set Absolute Position)	PSET
SR	(Report Configuration Status)	TSTAT
SS	(Report Configuration Status)	TSS
SSA	(RS-232 Echo Control)	ECHO
SSD	(Mode Alternate Stop Mode)	
SSF	(Normal/Low Velocity Range)	
SSG	(Clear/Save the Command Buffer On Limit)	COMEXL
SSH	(Clear/Save the Command Buffer on Stop)	COMEXS
SSI	(Enable/Disable Interactive Mode)	ERRLVL
SSJ	(Enable/Disable Continuous Scan Mode)	INSELP
SSL	(Enable Resume Execution)	COMEXS
SSN	(Set Message Mode)	ERRLVL
SSO	(Set Sequence Select)	

X Series Command List		6000 Series Command List
SSP	(Set Ratio Select)	
SSQ	(Set Drive Fault Polarity)	DRFLVL, INFEN
ST	(Shutdown)	DRIVE
STOP	(Stop)	S
STR	(Set Strobe Output Delay Time)	OUTPLC, OUTTW
T	(Time Delay)	T
TD	(Set Time Interval for Timed Data Streaming Mode)	STD
TDR	(Set Minimum Time Between Registration Moves)	INDEB
TEST	(Test Motion)	TEST
TF	(Set Following Time)	
TM	(Move Time Report)	TTIM
TR	(Wait For Trigger)	WAIT( ) and [IN]
TRD	(Read Timer from Parallel I/O)	T, [TW]
TS	(Report Trigger Status)	TIN
TW	(Set Thumbwheel Input Mode)	INPLC, INSTW, [TW]
TX	(Transmit Variable and String)	WRITE, WRVAR
U	(Pause and Wait for Continue)	!PS
UNTIL	(Until Part of REPEAT Command)	UNTIL( )
UR	(Report Scale Factor Status)	SCLD
US	(Set Position Scale Factor)	SCLD, SCALE
V	(Velocity)	V
VAR	(Variable)	VAR
VARD	(Read Variables via Parallel I/O)	VAR, [TW]
VARD	(Read Velocity via Parallel I/O)	[VEL], [VELA], [TW]
VS	(Start/Stop Velocity)	SSV
W1	(Signed Binary Position Report)	
W3	(Hexadecimal Position Report)	TPM
WHEN	(Set When Condition)	ERROR, ON
XBS	(Sequence Memory Available)	TMEM
XC	(Sequence Checksum Report)	
XD	(Sequence Definition)	DEF
XDIR	(Sequence Directory)	TDIR
XE	(Sequence Erase)	DEL
XEALL	(Erase all Sequences)	ERASE
XFK	(Set Fault or Kill Sequence)	ERRORP
XG	(Goto Sequence)	GOTO
XQ	(Sequence Interrupted Run Mode)	
XR	(Run a Sequence)	RUN
XRD	(Read Sequence via Parallel I/O)	INSELP
XRP	(Sequence Run with Pause)	RUN, PS
XS	(Sequence Execution Status)	TSS
XST	(Sequence Step Mode)	STEP
XT	(Sequence Termination)	END
XTR	(Set Trace Mode)	TRACE
XU	(Upload Sequence)	TPROG
XWHEN	(Set When Sequence)	ONP
Y	(Stop Loop)	LX
Z	(Reset)	RESET

Appendix C: ASCII Table

DEC	HEX	CHAR
0	00	NUL
1	01	SOH
2	02	STX
3	03	EXT
4	04	EOT
5	05	ENQ
6	06	ACK
7	07	BEL
8	08	BS
9	09	HT
10	0A	LF
11	0B	VT
12	0C	FF
13	0D	CR
14	0E	SO
15	0F	S1
16	10	DLE
17	11	XON
18	12	DC2
19	13	XOFF
20	14	DC4
21	15	NAK
22	16	SYN
23	17	ETB
24	18	CAN
25	19	EM
26	1A	SUB
27	1B	ESC
28	1C	FS
29	1D	GS
30	1E	RSt
31	1F	US
32	20	SPACE
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	`
40	28	(
41	29	)

DEC	HEX	CHAR
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	∅
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S

DEC	HEX	CHAR
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_
96	60	'
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
100	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}

DEC	HEX	CHAR
126	7E	~
127	7F	DEL
128	80	Ç
129	81	ü
130	82	é
131	83	â
132	84	ä
133	85	à
134	86	å
135	87	ç
136	88	ê
137	89	ë
138	8A	è
139	8B	ï
140	8C	î
141	8D	ì
142	8E	Ä
143	8F	Å
144	90	É
145	91	æ
146	92	Æ
147	93	ô
148	94	î
149	95	ò
150	96	û
151	97	ù
152	98	ÿ
153	99	Ö
154	9A	Ü
155	9B	¢
156	9C	£
157	9D	¥
158	9E	Pt
159	9F	*f*
160	A0	á
161	A1	í
162	A2	ó
163	A3	ú
164	A4	ñ
165	A5	Ñ
166	A6	ª
167	A7	º
168	A8	¿
169	A9	⌐
170	AA	¬
171	AB	½

DEC	HEX	CHAR
172	AC	¼
173	AD	¡
174	AE	«
175	AF	»
176	B0	▒
177	B1	▓
178	B2	▓
179	B3	¦
180	B4	┤
181	B5	╡
182	B6	╢
183	B7	╖
184	B8	╕
185	B9	╣
186	BA	║
187	BB	╗
188	BC	╝
189	BD	╜
190	BE	╛
191	BF	┐
192	C0	└
193	C1	┴
194	C2	┬
195	C3	├
196	C4	–
197	C5	†
198	C6	╞
199	C7	╟
200	C8	╚
201	C9	╔
202	CA	╩
203	CB	╦
204	CC	╠
205	CD	=
206	CE	╬
207	CF	╧
208	D0	╨
209	D1	╤
210	D2	╥
211	D3	╙

DEC	HEX	CHAR
212	D4	╘
213	D5	╒
214	D6	╓
215	D7	╫
216	D8	╪
217	D9	┘
218	DA	┌
219	DB	▓
220	DC	▄
221	DD	▌
222	DE	▐
223	DF	▀
224	E0	α
225	E1	β
226	E2	Γ
227	E3	π
228	E4	Σ
229	E5	σ
230	E6	∝
231	E7	τ
232	E8	Φ
233	E9	θ
234	EA	Ω
235	EB	δ
236	EC	∞
237	ED	∅
238	EE	∈
239	EF	∩
240	F0	≡
241	F1	±
242	F2	≥
243	F3	≤
244	F4	⌠
245	F5	⌡
246	F6	÷
247	F7	≈
248	F8	°
249	F9	•
250	FA	·
251	FB	√
252	FC	η
253	FD	2
254	FE	■
255	FF	

Index

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Y

Z

6000 Series Commands — *Functional Grouping*

ANI
ANIPOL
[ANI]
[FB]
[PANI]
[PCA]
TCA
TFB
TPANI
TPCA
TANI

Assignment & Comparison
[A]
[AD]
[ANI]
[ANV]
[AS]
[ASX]
[CA]
[CNT]
[D]
[DAC]
[DAT]
[DPTR]
[DREAD]
[DREADF]
[ER]
[FB]
[FS]
[IN]
[INO]
[LDT]
[LIM]
[MOV]
[NMCY]
[OUT]
[PANI]
[PC]
[PCA]
[PCC]
[PCE]
[PCL]
[PCM]
[PE]
[PER]
[PM]
[PMAS]
[PSHF]
[PSLV]
[READ]
[SEG]
[SS]
[TIM]
[TW]
[US]
[V]
[VAR]
[VARB]
[VEL]
[VELA]
[VMAS]

Branching – Conditional
ELSE
IF( )
NIF
NWHILE
REPEAT
UNTIL( )
WHILE( )

Branching – Unconditional
JUMP
GOSUB
GOTO
L
LN
LX
PLOOP
PLN

Command Buffer Control
COMEXC
COMEXK
COMEXL
COMEXP
COMEXR
COMEXS
GOWHEN

Command Delimiter
<cr>
<lf>
:

Communication Interface
[
]
ADDR
BOT
E
ECHO
EOL
EOT
ERRBAD
ERRDEF
ERRLVL
ERROK
FASTAT
PORT
[READ]
RESET
WRITE
WRVAR
WRVARB
WRVARS
XONOFF

Controller Configuration
CMDDIR
INDAX
INDUSE
INDUST
FASTAT
KDRIVE
MEMORY
PULSE
SFB

Counter
[CNT]
CNTE
CNTINT
CNTR

Data Storage
[DAT]
DATA
[DATP]
DATPTR
DATRST
DATSIZ
DATTCH
[DPTR]
TDPTR
[TW]

Display (RP240)
DCLEAR
DJOG
DLED
DPASS
DPCUR
[DREAD]
[DREADF]
DREADI
DRPCHK
DVAR
DWRITE

Drive Config.
DACTDP
DAREN
DAUTOS
DELVIS
DMTIND
DMTSTT
DRES
DRESET
DRFLVL
DRIVE
DWAVEF
INFEN
KDRIVE

Encoder
EMOVDB
ENC
ENCPOL
EPM
EPMDB
EPMG
EPMV
ERES
ESDB
ESK
ESTALL
[FB]
[PCE]
SFB
TFB
TPCE
TPE

Error Handling
[ER]
ERRBAD
ERRLVL
ERROR
ERRORP
TER

Feedrate Override
FR
FRA
FRH
FRL
FRPER

Following
FFILT
FMAXA
FMAXV
FMCLEN
FMCNEW
FMCP
FOLEN
FOLK
FOLMAS
FOLMD
FOLRD
FOLRN
FOLRNF
FOLSND
FPPEN
FSHFC
FSHFD
GOWHEN
[NMCY]
[PMAS]
[PSHF]
[PSLV]
SCLD
SCLMAS
TFS
TNMCY
TPMAS
TPSHF
TPSLV
TRGFN
TVMAS
[VMAS]

Homing
HOM
HOMA
HOMAA
HOMAD
HOMADA
HOMBAC
HOMDF
HOMEDG
HOMLVL
HOMV
HOMVF
HOMZ
[LIM]
TLIM

Inputs
ANVO
ANVOEN
[IN]
INDEB
INEN
INFEN
INFNC
INLVL
[INO]
INPLC
INSELP
INSTW
TIN
TINO
TRGFN

Interrupt to PC-AT
INTCLR
INTHW
INTSW
TINT

Jogging
JOG
JOGA
JOGAA
JOGAD
JOGADA
JOGVH
JOGVL

Joystick
[ANV]
ANVO
ANVOEN
[INO]
JOY
JOYA
JOYAA
JOYAD
JOYADA
JOYAXH
JOYAXL
JOYCDB
JOYCTR
JOYEDB
JOYVH
JOYVL
JOYZ
TANV
TINO

LDT
[FB]
[LDT]
LDTGRD
LDTPOL
LDTRES
LDTUPD
[PCL]
TFB
TLDT
TPCL

Limits (End-Of-Travel)
LH
LHAD
LHADA
LHLVL
[LIM]
LS
LSAD
LSADA
LSNEG
LSPOS
TLIM

Loops
L
LN
LX
PLOOP
PLN

Motion
A
[A]
AA
AD
[AD]
ADA
D
[D]
GO
GOWHEN
^K
K
MA
MC
[MOV]
PSET
S
SSV
TEST
V
[V]
[VEL]

Motion (Compiled)
FOLRNF
GOBUF
PCOMP
PRUN
PUCOMP
PLN
PLOOP
POUT
[SEG]
TSEG
VF

Motion (Linear Interpolated)
D
GOL
PA
PAA
PAD
PADA
PSCLA
PSCLV
PV
SCLD

Motion (S-Curve)
AA
ADA
HOMAA
HOMADA
JOGAA
JOGADA
JOYAA
JOYADA
LHADA
LSADA
PAA
PADA

On Condition (Program Interrupts)
ONCOND
ONIN
ONP
ONUS
ONVARA
ONVARB

Operators (Bitwise)
[&]
[|]
[^]
[~]
[<<]
[>>]

Operators (Logical)
[AND]
[NOT]
[OR]

Operators (Math)
[=]
[( )]
[+]
[-]
[*]
[/]
[SQRT]

Operators (Other)
!
@
;
$
#
'
[.]
["]
[\ ]
[
]

Operators (Relational)
[=]
[>]
[>=]
[<]
[<=]
[<>]

Operators (Trig)
[ATAN( )]
[COS( )]
[PI]
RADIAN
[SIN( )]
[TAN( )]

Outputs
OUT
[OUT]
OUTALL
OUTANA
OUTEN
OUTFEN
OUTFNC
OUTLVL
OUTPA
OUTPB
OUTPC
OUTPD
OUTPLC
OUTTW
TOUT

Path Contouring
DEF
END
MEMORY
PA
PAA
PAB
PAD
PADA
PARCM
PARCOM
PARCOP
PARCP
PAXES
PCOMP
PL
PLC
PLIN
POUT
PPRO
PRTOL
PRUN
PSCLA
PSCLD
PSCLV
PTAN
PUCOMP
PV
PWC
TDIR
TMEM

Power-Up Execution
STARTP

Program Debug Tools
#
ANVO
ANVOEN
BP
HELP
INEN
OUTEN
STEP
TCMDER
TRACE
TRANS

Program Definition & Execution
DEF
DEL
END
ERASE
ERRORP
INFNC
INSELP
MEMORY
ONP
RUN
[SEG]
STARTP
TDIR
TEX
TMEM
TSEG
$

Program Flow Control
BP
BREAK
C
ELSE
GOSUB
GOTO
GOWHEN
HALT
IF( )
JUMP
L
LN
LX
NIF
NWHILE
PS
REPEAT
T
UNTIL( )
WAIT( )
WHILE( )

Registration & Position Capture
INDEB
INFNC
RE
REG
REGLOD
REGSS
[PCA]
[PCC]
[PCE]
[PCL]
[PCM]
TPCA
TPCC
TPCE
TPCL
TPCM

Scaling
PSCLA
PSCLD
PSCLV
SCALE
SCLA
SCLD
SCLV

Servo
CMDDIR
[DAC]
DACLIM
DACMIN
[FB]
OUTANA
SDTAMP
SDTFR
SFB
SGAF
SGAFN
SGENB
SGI
SGIN
SGILIM
SGP
SGPN
SGSET
SGV
SGVF
SGVFN
SGVN
SMPER
SOFFS
SOFFSN
SSFR
SSWD
SSWG
STRGTD
STRGTE
STRGTT
STRGTV
TFB
TSTLT
TVELA

Streaming
SD
STD
STREAM

Subroutine Definition & Execution
DEF
DEL
END
ERASE
GOSUB
GOTO
JUMP
MEMORY
RUN
$

Timer
[TIM]
TIMINT
TIMST
TIMSTP

Transfers (Status)
TANI
TANV
TAS
TASF
TASX
TASXF
TCA
TCMDER
TCNT
TDAC
TDIR
TDPTR
TER
TERF
TEX
TFB
TFS
TFSF
TGAIN
TIN
TINO
TINOF
TINT
TLABEL
TLDT
TLIM
TMEM
TNMCY
TOUT
TPANI
TPC
TPCA
TPCC
TPCE
TPCL
TPCM
TPE
TPER
TPM
TPMAS
TPROG
TPSHF
TPSLV
TREV
TSEG
TSS
TSSF
TSTAT
TSTLT
TTIM
TUS
TVEL
TVELA
TVMAS

Variables
VAR
[VAR]
VARB
[VARB]
VARCLR
VARS
VCVT( )

[ ] = Command is used for assignment and/or comparison operations.

6000 Series Software Reference

General Programming

Overview of command description format in this manual — see page 2.

Programming Guidelines:
- Command syntax — see pages 3 & 4.
- Command value substitutions — see page 5.
- Programmable I/O bit patterns for all 6000 products — see page 6.
- Error Messages and identifying bad commands — see pages 7-9
- Other guidelines — see *6000 Series Programmer's Guide*.

Command-to-product compatibility — see Appendix A.
6000 vs. X Language compatibility — see Appendix B.
List of commands: Functional list found on inside of this back cover.
Alphabetical list found in Appendix A.

Use Motion Architect:
- Create a setup program (Setup feature).
- Create and edit programs (Editor feature).
- Download and upload programs, check status (Terminal feature).
- Set up custom test/status displays (Panel feature).
- Programming Tip: Resize the Editor and Terminal windows to be side by side; this helps you quickly jump back and forth between editing and downloading programs and checking responses.
- From the on-line "6000 Software Reference" (Help menu), look up command data and copy/paste code samples into your program.

Common Problems

Detailed troubleshooting instructions — refer to the *6000 Series Programmer's Guide*:
- Problem/Cause/Remedy table
- Program debug tools (error messages, Trace, Single-Step, etc.)
- Downloading error table (bus-based products)

Common problems:
- No motion. Possible causes include:
  - P-CUT input or ENBL input must be grounded to allow motion. Check status with TINOF command (see bit #6).
  - Drive is disabled. Enable with DRIVE command. Make sure the drive fault level (DRFLVL) is correct.
  - End-of-travel limit is active (check with TASF, bits 15-18). If you are NOT using hardware limit switches, disable them with the @LH0 command.
- Programmable input (INFNC) functions do not work, and drive fault detection does not work:
  - Enable input functions with the INFEN1 command.
- Scaled values are incorrect:
  - See Scaling section in the *6000 Series Programmer's Guide*.

Technical Assistance: Phone numbers are reported with the HELP command, and are listed on the inside cover of this document.

Programming Scenario

WARNING This program executes motion. If you coupled the motor to the load, make sure it is safe to move the load without damaging equipment or injuring personnel. Be ready to enter the ctrl/K command (from the terminal) to "kill" motion.

NOTE: A more detailed/illustrated version of this programming scenario is provided in Chapter 1 of the *6000 Series Programmer's Guide*.

① Install your 6000 Series product according to the instructions in your product's *Installation Guide*. If you have a servo product, you should tune your system to ensure that you get predictable motion.

② Install Motion Architect (disks are in the ship kit) and then launch Motion Architect.

③ From the Motion Architect main menu, click on "Product", click on "Selection" and select the name of the product you are using. Click **OK**.

④ From the Motion Architect main menu, click on "Editor" to launch Motion Architect's program editor.

⑤ In the Editor, type in the program as shown below. The comments (to the right of ";") are shown here to explain the function of each command; you do not have to type them in. This is a program for single-axis motion.

```
DEL EXAMPL ;Delete the program called EXAMPL
DEF EXAMPL ;Begin defining the program EXAMPL
DRIVE1     ;Enable the drive
MC0        ;Select preset positioning mode
MA0        ;Select incremental positioning mode
LH3        ;Enable hard end-of-travel limits
           ;<<WARNING>> If you are not using
           ;hardware end-of-travel limit inputs,
           ;you can instead use the LH0 command
           ;to disable the limits, but USE CAUTION
           ;so that you do not move the load too
           ;far in either direction.
A25        ;Set acceleration to 25 units/sec/sec
AD25       ;Set deceleration to 25 units/sec/sec
V5         ;Set velocity to 5 units/sec
D16000     ;Set distance to 16000 units
           ;<<WARNING>> Make sure this is a safe
           ;distance if you disabled the end-of-
           ;travel limits (LH0 command) above.
GO         ;Execute the move
END        ;End definition of program EXAMPL
```

⑥ In the Editor, save the program as "exampl.prg".

⑦ (If have not already done so, power up your 6000 Series product.) From the Motion Architect main menu, click on "Terminal" to launch Motion Architect's terminal emulator. From the "Transfers" menu, select "Send Motion Program(s)", choose the "exampl.prg" file, and click **OK**.

⑧ Run the Program: To run the program from the terminal emulator, type the name of the program (in this case it is EXAMPL) and press the carriage return or enter key.

If you are using a stepper, the default resolution will most likely be set to 25,000 steps per rev; therefore, running the EXAMPL program should move the motor about 2/3 rev.

If you are using a servo, the default resolution will most likely be set to 4,000 counts per rev; therefore, running the EXAMPL program should move the motor about 4 revs.

Status Commands

TASF — Current axis-specific conditions

1.....Axis in motion
2.....Commanded direction negative
3.....Accelerating
4.....At commanded velocity
5.....Home successful (HOM)
6.....In Absolute Positioning Mode (MA)
7.....In Continuous Positioning Mode (MC)
8.....In Jog Mode (JOG)
9.....In Joystick Mode (JOY)
10...In Encoder Step Mode (ENC)
11...Position Maintenance on (EPM)
12...Stall was detected (ESTALL)
13...Drive shutdown occurred
14...Drive fault occurred
15...POS hardware limit hit
16...NEG hardware limit hit
17...POS software limit (LSPOS) hit
18...NEG software limit (LSNEG) hit
19...Within Deadband (EPMDB)
20...In Position (COMEXP)
21...In Distance Streaming Mode
22...In Velocity Streaming Mode
23...Position error limit (SMPER) was exceeded
24...Load is within the Target Zone (STRGTD & STRGTV)
25...Target Zone timeout (STRGTT)
26...Motion pending due to GOWHEN
27...LDT position read error
28...Registration move occurred
29...RESERVED
30...Pre-emptive (on-the-fly) GO or Registration move not possible
31...RESERVED
32...RESERVED

TSSF — Current system conditions

1.....System is ready
2.....RESERVED
3.....Executing a program
4.....Last command executed was immediate
5.....In ASCII Mode
6.....In Echo Mode (ECHO)
7.....Defining a program (DEF)
8.....In Trace Mode (TRACE)
9.....In Step Mode (STEP)
10...In Translation Mode
11...Command error occurred
12...Break point (BP) is active
13...Pause active (PS or pause input)
14...Wait (WAIT) is active
15...Monitoring ONCOND conditions
16...Waiting for data (READ)
17...Loading thumbwheel data (TW)
18...Program Select Mode (INSELP)
19...Dwell is progress (T)
20...Waiting for RP240 data (DREAD or DREADF)
21...RP240 is connected
22...Non-volatile memory error
23...Gathering servo data
24...RESERVED
25...Position captured with TRG-A
26...Position captured with TRG-B
27...Position captured with TRG-C
28...Position captured with TRG-D
29...Compiled memory is 75% full
30...Compiled memory is 100% full
31...Compile (PCOMP) failed
32...RESERVED

Other Status Commands *(subset)*

TSTAT.....General system status.
TCMDER.....Identifies the command that caused an error.
TEX.....Execution status of the current program in progress.
TDIR.....Names of all stored programs; memory usage.
TFSF.....Following status.
TERF.....Error status. You must first enable each error-checking bit with the ERROR command. If the error-checking bit is enabled and the error occurs, the controller will branch to the error program assigned with the ERRORP command.
TINOF.....Status of P-CUT or ENBL input and joystick inputs.
INFNC.....Function and active status of each programmable & trigger input. Bit assignments vary by product—refer to page 6.
OUTFNC.....Function and active status of each programmable & auxiliary output. Bit assignments vary by product—refer to page 6.
TPM.....Commanded position (steppers).
TPC.....Commanded position (servos).
TFB.....Actual position, selected feedback sources (servos).
TPER.....Position error (difference of commanded vs. actual).
TASXF.....Extended axis status.

On-line Documentation

The contents of this document may be viewed and printed from the PDF files on our web site (www.compumotor.com) and from the hypertext version accessed via the Help menu in Motion Architect.