ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

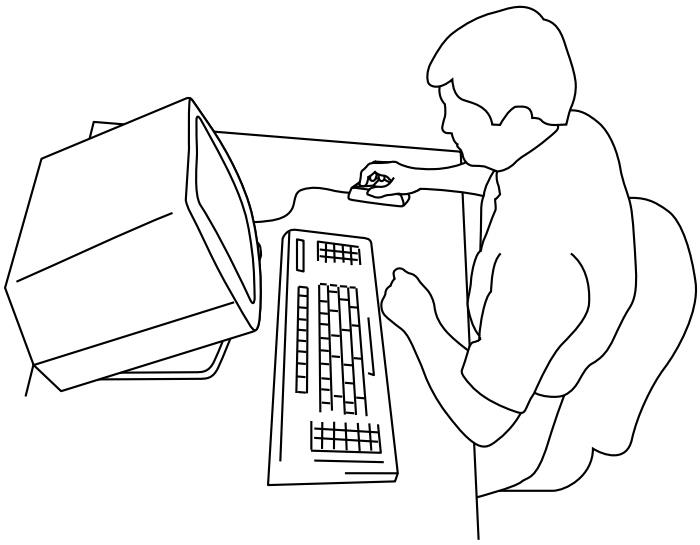
Expert team | Trust guarantee | 100% satisfaction

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Find the *Parker / Compumotor ZETA6104* at our website: *Click HERE*

Compumotor

6000 Series
Programmer's Guide



Compumotor Division
Parker Hannifin Corporation
p/n 88-014540-01

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

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.

6000 Series products and the information in this user guide are the proprietary property of Parker Hannifin Corporation or its licensers, and may not be copied, disclosed, or used for any purpose not expressly authorized by the owner thereof.

Since Parker Hannifin constantly strives to improve all of its products, we reserve the right to change this user guide and software and hardware mentioned therein at any time without notice.

In no event will the provider of the equipment be liable for any incidental, consequential, or special damages of any kind or nature whatsoever, including but not limited to lost profits arising from or in any way connected with the use of the equipment or this user guide.

© 1991-97, Parker Hannifin Corporation
All Rights Reserved

Motion Architect is a registered trademark, and Motion Builder, Servo Tuner, Motion OCX Toolkit, CompuCAM and DDE6000 are trademarks of Parker Hannifin Corporation.
Microsoft and MS-DOS are registered trademarks, and Windows, DDE, Visual Basic, and Visual C++ are trademarks of Microsoft Corporation.
Wonderware is a registered trademark, and InTouch and NetDDE are trademarks of Wonderware Corporation.
Motion Toolbox is a trademark of Snider Consultants, Inc.
LabVIEW is a registered trademark of National Instruments Corporation.

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

North America and Asia:
Compumotor Division of Parker Hannifin
5500 Business Park Drive
Rohnert Park, CA 94928
Telephone: (800) 358-9070 or (707) 584-7558
Fax: (707) 584-3793
FaxBack: (800) 936-6939 or (707) 586-8586
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

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Change Summary

6000 Series Programmer's Guide

Rev D

Rev D – summary of changes (minor)

- Removed references to the ***Motion Architect User Guide***. Motion Architect no longer ships with the printed manual. An on-line Adobe Acrobat PDF version is available from our web site (http://www.compumotor.com).
- Clarified *Command Value Substitution* guidelines with a "rule of thumb" (see page 6).
- Error programming correction: DO NOT use the `ERRORP CLR` command to cancel the branch to the error program (see page 31).
- DLL documentation clarification and correction (see pages 51-62):
  - WN956000.DLL is a 32-bit DLL to be used for Windows 95.
  - The DLL functions SetNTParam, SetNTMultiCardAddress, and SetDevice return FALSE ("0") if the function is successful and TRUE (non-zero value) if the function is unsuccessful.
- Sending `ERRLVL1` to the first unit in an RS-232 daisy-chain does not set all other units to `ERRLVL1`. You must send `ERRLVL1` to each unit individually. (see page 72)
- `GOWHEN` correction: A preset `GO` command that is already in motion can (not "cannot") start a new profile using the `GOWHEN` and `GO` sequence of commands. (this is true as of rev 4.1 firmware)

Rev C – summary of changes (MAJOR)

- Updated to accommodate 4.x firmware enhancements (see topics below) and the ZETA6104.
- 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.
- Added documentation to support all 6000 products.
- Incorporated the *Following User Guide* (see Chapter 6).
- New sections:
  - *Programming Scenario*.....Page 8
  - *Controlling Multiple Serial Ports*.....Page 70
  - *RS-485 Multi-drop*.....Page 75
  - *Setup Parameters* (list of commands for common setup parameters).....Page 78
  - *ZETA610n Internal Drive Setup*.....Page 82
  - *Servo Setup*.....Page 98
  - *RP240 Remote Operator Panel*.....Page 130
  - *Host Computer Interface*.....Page 143
  - *Graphical User Interface (GUI) Development Tools*.....Page 144
  - *Compiled Motion Profiling*.....Page 163
  - *On-the-Fly Motion*.....Page 178
  - *Registration*.....Page 182
  - *Synchronizing Motion*.....Page 186
  - Chapter 6: *Following* (incorporated the *Following User Guide*).....Page 191
  - *Status Commands*.....Page 232
- New back cover with quick-reference material.

READ ON ...
for a summary of the enhancements implemented in firmware revision 4.x.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Topic	Description
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. (SEE PG. 97 OR 101)
Compiled Motion	**New Feature**: (SEE PG. 163) 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, contouring is now available for all multi-axis products, steppers and servos. (SEE PG. 153)
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." (SEE PG. 178) 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: (SEE PG. 82) DACTDP ....Enable/disable active damping for speeds greater than 3 rps. (config. 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. (config. procedure: see the *ZETA6104 Installation Guide*) DAUTOS ....Enable/disable automatic current standby mode in which current to the motor (& torque) 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. (SEE PG. 97 OR 100)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Topic	Description
Error Checking Conditions	**Enhancements**: (SEE PG. 31) • 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 (TASX 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 - 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. - Error bit #16 is set if a bad command was detected; clear with TCMDER. • Related commands: [ER]..... Error Status (assignment or comparison) ERROR ..... Error-Checking Enable ERRORP.... Error Program Assignment TER..... Transfer Error Status
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. (SEE PG. 43) **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 input. Servo products only. • 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. (SEE PG. 166) • The *Following User Guide* has been incorporated into this document (SEE PG. 192).
Homing	**Clarification**: Avoid using pause and resume functions during the homing operation. 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 & 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). (SEE PG. 12) • These commands are automatically saved in non-volatile memory: (SEE PG. 33) CMDDIR.... Commanded Direction Polarity (6104, 615n, 620n, 6270 only) DMTIND.... Motor Inductance (6104 only) DMTSTT.... Motor Static Torque (6104 only) DRPCHK.... RP240 check (6104, 615n, 620n, & 625n only) ENCPOL.... Encoder Polarity (6104, 620n, & 6270 only)
On-The-Fly Motion (AKA: *Pre-Emptive GOs*)	**Enhancements**: (SEE PG. 178) • 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. *Continued on next page*

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Topic	Description
On-The-Fly Motion *(continued)*	• 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**: (SEE PG. 182) • 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 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 Ø) 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 & ER) and branch to the assigned ERRORP error-handling program. Axis status bit #30 and error status bit #10 are cleared (set to Ø) 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**: (SEE PG. 70) • 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. • 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
Servo Updates Changed (Servo Products Only)	*(see SSFR command description for full explanation of table contents)*

# of Axes (INDAX)	SSFR 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)
1	1	3030	330	3030	330	757	1320
1	2	5405	185	2703	370	675	1480
Default, Single-Axis 1	4	6250	160	1563	640	520	1920
1	8	6667	150	833	1200	417	2400
2	1	2353	425	2352	425	588	1700
2	2	3571	280	1786	560	446	2400
Default, Two-Axis 2	4	3571	280	893	1120	446	2400
2	8	3571	280	446	2240	446	2400
3	1	1667	600	1667	600	555	1800
3	2	2222	450	1111	900	555	1800
3	4	2353	425	588	1700	588	1700
4	1	1250	800	1250	800	417	2400
4	2	1667	600	833	1200	417	2400
Default, Four-Axis 4	4	2000	500	500	2000	500	2000

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Topic	Description
Status Reporting	**Enhancements**: (SEE PG. 232) • New transfer (display status) commands: TASX ..... Transfer extended axis status. Bit assignments are as follows: Bit #1: Motor fault (6104 only) Bit #2: Drive low voltage fault (6104 only) Bit #3: Drive over-temperature fault (6104 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. • 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 not affected by scaling. The VELA assignment/comparison operator for TVELA is now available as of rev 4.0.
Target Zone	• 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 all 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. (SEE PG. 105) • Target Zone Commands: STRGTE.... Target Zone Mode Enable/Disable STRGTD.... Target Distance Zone STRGTT.... Target Settling Timeout Period STRGTV.... Target Velocity Zone

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

New Commands in Revision 4.x *(including product compatibility)*

Command	Name	AT6200	AT6400	AT6250	AT6450	610n	615n	620n	625n	6270
[	Send Response to All Ports					X	X	X	X	X
]	Send Response to Alternate Port					X	X	X	X	X
ASX	Extended Axis Status	X	X	X	X	X	X	X	X	X
BOT	Beginning of Transmission Characters	X	X	X	X	X	X	X	X	X
DACTDP	Active Damping					X				
DAREN	Anti-Resonance					X				
DAUTOS	Auto Current Standby					X				
DELVIS	Electronic Viscosity					X				
DMTIND	Motor Inductance					X				
DMTSTT	Motor Static Torque					X				
DRESET	Drive Reset						X			
DRPCHK	Remote Port Check					X	X	X	X	X
DWAVEF	Waveform					X				
FOLK	Following Kill			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 & Direction	S	S	X	X	X	X	X	X	X
GOBUF	Store a Motion Segment in a Buffer	X	X	X	X	X	X	X	X	X
PCOMP *	Compile a Program	X	X	X	X	X	X	X	X	X
PLN	Loop End, Compiled Motion	X	X	X	X	X	X	X	X	X
PLOOP	Loop Start, Compiled Motion	X	X	X	X	X	X	X	X	X
PORT	Designate Communications Port					X	X	X	X	X
POUTA	Output on Axis 1, Compiled Motion	X	X	X	X	X	X	X	X	X
POUTB	Output on Axis 2, Compiled Motion	X	X	X	X			X	X	X
POUTC	Output on Axis 3, Compiled Motion		X		X					
POUTD	Output on Axis 4, Compiled Motion		X		X					
PRUN *	Run a Compiled Program	X	X	X	X	X	X	X	X	X
PUCOMP *	Un-Compile a Program	X	X	X	X	X	X	X	X	X
REGLOD	Registration Lock-Out Distance	X	X	X	X	X	X	X	X	X
REGSS	Registration Single Shot	X	X	X	X	X	X	X	X	X
[SEG]	Number of Free Segment Buffers	X	X	X	X	X	X	X	X	X
TASX	Transfer Extended Axis Status	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
[VELA]	Velocity (Actual) Assignment	S	S	X	X	X	X	X	X	X
VF	Final Velocity	X	X	X	X	X	X	X	X	X
XONOFF	Enable/Disable XON/XOFF					X	X	X	X	X

* *Modified to support compiled motion (previously, these commands supported only path contouring).*
S *Applicable only to the standard (not OEM) version of the product.*

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

TABLE OF CONTENTS

Overview

About This Manual i
- Organization of This Manual i
- Programming Examples ii
- Reference Documentation ii
- Assumptions of Technical Experience ii
- Product Name References (What's in a Name?) iii

Before You Begin iii
Support Software iii
- Motion Architect iii
- Motion Builder iv
- Motion Toolbox iv
- DOS Support Software iv

Technical Support iv

Chapter 1. Programming Fundamentals

Motion Architect Programming Environment 2
- Side-by-Side Editor and Terminal Windows 2

Command Syntax 3
- Overview 3
- Description of Syntax Letters and Symbols 4
- General Guidelines for Syntax 5
- Command Value Substitutions 6
- Assignment and Comparison Operators 6
- Programmable Inputs and Outputs Bit Patterns 8

Creating Programs 8
- Program Development Scenario 8

Storing Programs 12
- Storing Programs in Stand-Alone Products 12
- Storing Programs in Bus-Based Products 12
- Memory Allocation 12
- Checking Memory Status 13

Executing Programs (options) 14
Creating and Executing a Set-up Program 14
- Set-up Program Execution *for Stand-Alone Controllers* . 15
- Set-up Program Execution *for Bus-Based Controllers* . . 15

Program Security 15
Controlling Execution of Programs and the Command Buffer 16
- Continuous Command Execution 16
- Continue Command Execution on Kill 16
- Save Command Buffer on Limit 17
- Pause Command Execution Until In Position Signal 17
- Effect of Pause/Continue Input 17
- Save Command Buffer on Stop 17

Restricted Commands During Motion 18
Variables 18
- Converting Between Binary and Numeric Variables . 19
- Using Numeric Variables 19
- Using Binary Variables 22

Program Flow Control 23
- Unconditional Looping and Branching 23
- Conditional Looping and Branching 24

Program Interrupts (ON Conditions) 29
Error Handling 30
- Enabling Error Checking 30
- Defining the Error Program 30
- Canceling the Branch to the Error Program 31
- Error Program Set-up Example 32

Non-Volatile Memory (Stand-Alone Products Only) 33
System Performance 33

Chapter 2. Communication

Motion Architect Communication Features 36
DOS Support Software for Stand-Alone Products 37
DOS Support for Bus-Based Products 38
- Downloading the Operating System 39
- Terminal Emulation 41
- Downloading Application Programs from the DOS prompt 41
- Creating Your Own DOS-Based Application Program. 42

PC-AT Bus Communication Registers 43
- Fast Status Register (Base+2, Base+3) 43
- Card Status and Interrupts to/from PC-AT (Base+4) 49
- Reading and Writing to the 6000 Controller 50

DDE6000 50
DLLs 51
- Visual Basic™ Support 52
- Visual C++™ Support 58

Motion OCX Toolkit™ 62
PC-AT Interrupts 63
- AT6nnn Interrupt Path 63
- How to Use Interrupts 65
- Interrupt-Driven Terminal Emulator 68

Controlling Multiple Serial Ports 70
- Configuring the COM Port 70
- Selecting a Destination Port for Transmitting from the Controller 71

RS-232C Daisy-Chaining 72
- Daisy-Chaining from a Computer or Terminal 73
- Daisy-Chaining from a Master 6000 Controller 74
- Daisy-Chaining and RP240s 74

RS-485 Multi-Drop 75

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Chapter 3. Basic Operation Setup

Before You Begin....78
Setup Parameters Discussed in this Chapter....78
Using a Setup Program....79
Motion Architect....79
Resetting the Controller....79
Participating Axes....79
Memory Allocation....80
Drive Setup....80
Drive Fault Level....80
Drive Resolution (steppers only)....81
Step Pulse (steppers only)....81
Start/Stop Velocity (steppers only)....82
Disable Drive On Kill (servos only)....82
ZETA610n Internal Drive Setup....82
Axis Scaling....83
When Should I Define Scaling Parameters?....83
Acceleration & Deceleration Scaling (SCLA and PSCLA)....84
Velocity Scaling (SCLV and PSCLV)....84
Distance Scaling (SCLD and PSCLD)....85
Scaling Examples....86
Positioning Modes....87
Preset Positioning Mode....88
Continuous Positioning Mode....89
End-of-Travel Limits....90
Homing....91
Closed-Loop Stepper Setup (steppers only)....95
Encoder Resolution....95
Encoder Step Mode....95
Position Maintenance....95
Position Maintenance Deadband....95
Stall Detection & Kill-on-Stall....96
Stall Deadband....96
Encoder Set Up Example....96
Use the Encoder as a Counter....96
Encoder Polarity....97
Commanded Direction Polarity....97
Servo Setup....98
Tuning....99
Feedback Device Polarity....100
Commanded Direction Polarity....101
Dither....101
DAC Output Limits....102
Servo Control Signal Offset....102
Servo Setup Examples....103
Target Zone Mode....105
Programmable Inputs and Outputs (including triggers and auxiliary outputs)....106
Programmable I/O Bit Patterns....107
Input Functions....108
Output Functions....116
Variable Arrays (teaching variable data)....120
Basics of Teach-Data Applications....120
Summary of Related 6000 Series Commands....122
Teach-Data Application Example....122

Chapter 4. User Interface Options

Safety Features....126
Options Overview (application examples)....127
Stand-Alone Interface Options....127
Programmable Logic Controller....127
Host Computer Interface....127
Custom Graphical User Interfaces (GUIs)....127
Programmable I/O Devices....128
Programmable I/O Functions....128
Thumbwheels....129
PLCs....130
RP240 Remote Operator Panel....130
Configuration....131
Operator Interface Features....131
Using the Default Menus....132
Joystick and Analog Inputs....138
Joystick Control....138
Feedrate Override *(multi-axis steppers only)*....141
ANI Analog Input Interface *(products with ANI option)*....142
Auxiliary Analog Output (“half axis” — AT6n50 only)....142
Host Computer Interface....143
Graphical User Interface (GUI) Development Tools....144

Chapter 5. Custom Profiling

S-Curve Profiling (servos only)....146
Timed Data Streaming (bus-based steppers only)....148
Time-Distance Streaming Example....149
Linear Interpolation....152
Contouring (Circular Interpolation)....153
Path Definition....153
Participating Axes....154
Path Acceleration, Deceleration, and Velocity....155
Segment End-point Coordinates....155
Line Segments....156
Arc Segments....157
Segment Boundary....158
Using the C Axis (4-axis products only)....159
Using the P Axis (4-axis products only)....159
Outputs Along the Path....160
Paths Built Using 6000 Series Commands....160
Compiling the Path....160
Executing the Path....161
Possible Programming Errors....161
Programming Examples....161
Compiled Motion Profiling....163
Compiled Following Profiles....166
Dwells and Direction Changes....168
Compiled Motion Versus On-The-Fly Motion....169
Related Commands....169
Compiled Motion — Sample Applications....170
On-the-Fly Motion (pre-emptive GOs)....178
OTF Error Conditions....179
OTF Sample Application....180
Registration....182
Registration Move Accuracy....182
Preventing Unwanted Registration Moves....182
Registration Move Status & Error Handling....183
How to Set up a Registration Move....183
Registration – Sample Applications....184
Synchronizing Motion....186
Conditional “GO”s (GOWHEN)....186
Trigger Functions (TRGFN)....189

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Chapter 6. Following

Ratio Following – Introduction 192
- What can be a master? 192
- Performance Considerations 193
- Following Status 193

Implementing Ratio Following 194
- Ratio Following Setup Parameters 194
- Slave vs. Master Move Profiles 199
- Performing Phase Shifts 201
- Summary of Ratio Following Commands 203
- Electronic Gearbox Application 204
- Trackball Application 205

Master Cycle Concept 207
- Master Cycle Commands 207
- Summary of Master Cycle and Wait Commands 210
- Continuous Cut-to-Length Application 211

Technical Considerations for Following 213
- Master Position Prediction 214
- Master Position Filtering 214
- Following Error 215
- Maximum Velocity and Acceleration 216
- Dynamic Position Maintenance 216
- Factors Affecting Following Accuracy 216
- Preset vs. Continuous Following Moves 219
- Master and Slave Distance Calculations 220
- Using Other Features with Following 221

Troubleshooting for Following 223

Following Commands 225

Chapter 7. Troubleshooting

Troubleshooting Basics 228

Solutions to Common Problems 228

Program Debug Tools 231
- Status Commands 232
- Error Messages 236
- Trace Mode 239
- Single-Step Mode 240
- Simulating I/O Activation 240
- Simulating Analog Input Channel Voltages 242
- Motion Architect's Panel Module 242

Downloading Error Table (bus-based controllers only) 243

Technical Support 244

Product Return Procedure 244

Index 245

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

O V E R V I E W

About This Manual

This manual is designed to help you implement the 6000 Series Product's features in your application. Detailed feature descriptions are provided, including application scenarios and programming examples. For details on each 6000 Series command, refer to the ***6000 Series Software Reference***.

Organization of This Manual

The feature descriptions are grouped into chapters as listed below.

Chapter	Information
Chapter 1. *Programming Fundamentals*	Discussion of essential programming guidelines and standard programming features such as branching, variables, interrupts, error handling, etc.
Chapter 2. *Communication*	Communication considerations, such as using Motion Architect, DDE, DLL and OCX tools, bus communication registers, PC-AT interrupts, DOS Support Disk communication files, RS-232 daisy-chains and RS-485 multi-drops, etc.
Chapter 3. *Basic Operation Setup*	General operation setup conditions, such as number of axes, scaling factors, feedback device setup, end-of-travel limits, homing, etc.
Chapter 4. *User Interface Options*	Considerations for implementing various user interfaces such as programmable I/O, a joystick, an RP240, etc.
Chapter 5. *Custom Profiling*	Descriptions of custom profiling features such as S-Curves, timed data streaming, linear and circular interpolation, compiled profiles, on-the-fly motion profiling, registration, and synchronized motion.
Chapter 6. *Following*	Feature descriptions and application examples for using Following features.
Chapter 7. *Troubleshooting*	Methods for isolating and resolving hardware and software problems.

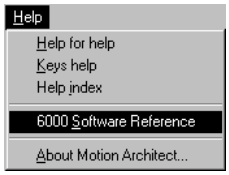
User Guide Feedback Please send your comments to our email address: `6000user@cmotor.com`

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Programming Examples

Programming examples are provided to demonstrate how the 6000 product's features may be implemented. These examples are somewhat generalized, due to the diverse nature of the family of 6000 Series products and their application; consequently, some attributes, such as the number of axes used or the I/O bit pattern referenced, may differ from those available with your particular 6000 product.

Additional sample programs can be found in the APP_PRGS sub-directory of the Motion Architect directory (MA6000\APP_PRGS). These files may be opened and edited in Motion Architect's Editor module, then downloaded using Motion Architect's Terminal module.



TIP: From the Help menu in Motion Architect and from our web site (www.compumotor.com), you can access the online versions of this *Programmer's Guide* and the *Software Reference*. You can copy the programming examples from these online documents and paste them into Motion Architect's Program Editor. Then you can edit the code for your application requirements and download the program using the Terminal Emulator. For additional tips on using the Editor and Terminal, refer to page 2 in this manual or to the ***Motion Architect User Guide.***

Reference Documentation

This document is intended to accompany the documents listed below, as part of the 6000 product user documentation set.

ONLINE ACCESS:

Online versions of this Programmer's Guide and the Software Reference are available from the Help menu in Motion Architect.

INTERNET ACCESS:

These documents are also available to view and print from our web site (www.compumotor.com).

Reference Document	Information
6000 product installation guide	Hardware-related information specific to the 6000 Series product. • Product description • Installation instructions • Drive information (packaged controller/drive products only) • Hardware reference • Troubleshooting procedures • Servo tuning (procedures for tuning without Servo Tuner™) • Electrical noise reduction techniques
6000 Series Software Reference *	Detailed descriptions of all 6000 Series commands. Quick-reference tables are also provided: • Product-to-command compatibility table • X-to-6000 language compatibility table • ASCII table
Motion Architect User Guide	Overview and user tips for Motion Architect features, and guidelines for using the dynamic link library (DLL). **This manual is only available in Acrobat PDF format from our web site.**
RP240 User Guide	Detailed user instructions for the RP240 remote operator panel (optional peripheral device for serial based products only).
User guides for optional software tools:	☞ ***Motion Builder Startup Guide*** ☞ ***Servo Tuner User Guide*** ☞ ***CompuCAM User Guide*** ☞ ***Motion Toolbox User Guide*** ☞ ***Motion OCX Toolkit User Guide***

* Also available as an on-line hypertext utility, accessed from the Help menu in Motion Architect.

Assumptions of Technical Experience

To effectively use the information in this manual, you should have a fundamental understanding of the following:

- Electronics concepts such as voltage, switches, current, etc.
- Motion control concepts such as motion profiles, torque, velocity, distance, force, etc.
- Programming skills in a high-level language such as C, BASIC, or Pascal is helpful
- IBM/compatible bus architecture and communication protocol (bus-based products only)
- If you are new to the 6000 Series Programming Language, read Chapter 1 thoroughly.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Product Name References (What's in a Name?)

This document sometimes uses one product name to reference an entire subset of the 6000 family (e.g, "AT6n50" refers to the AT6450 and the AT6250; "AT6n00" refers to the AT6200 and the AT6400). Unless otherwise noted, references to a standard product are applicable to the OEM version as well (e.g., AT6400 and OEM-AT6400; 6200 and OEM6200).

Before You Begin

Before you begin to implement the 6000 controller's features in your application you should complete the items listed below.

- Complete all the installation and test procedures provided in your 6000 product's ***Installation Guide***.
- If you are using a servo control product, complete the tuning procedures. If you are using Servo Tuner, use the instructions in the ***Servo Tuner User Guide***. If you are using an empirical tuning method (not Servo Tuner), refer to the procedures provided in the *Tuning* appendix of the product's ***Installation Guide***.
- Keep the ***6000 Series Software Reference*** close at hand to answer questions about specific 6000 Series commands. If you are new to the 6000 Series Programming Language, read Chapter 1 (*Programming Fundamentals*) thoroughly.

Support Software

These software development tools are available to help you program your 6000 Series product.

- Motion Architect, and these add-on modules:
  - Servo Tuner (tuning and data capture)
  - CompuCAM (CAD-to-Motion)
- Motion Builder (graphical icon-based programming software)
- Motion Toolbox (Motion VIs for LabVIEW)
- DOS Support Software

Motion Architect®

All 6000 Series products are shipped with Motion Architect, an intuitive Microsoft® Windows™ based programming tool. A brief description of Motion Architect's basic features is provided below. For more detailed user information, refer to the ***Motion Architect User Guide***.

- **System Configurator and Code Generator**: Automatically generate controller code of basic system set-up parameters (I/O definitions, feedback device operations, etc.).
- **Program Editor**: Create blocks or lines of 6000 controller code, or copy portions of code from previous files. You can save program editor files for later use in BASIC, C, etc., or in the terminal emulator or test panel.
- **Terminal Emulator**: Communicating directly with the 6000 controller, the terminal emulator allows you to type in and execute controller code, transfer code files to and from the 6000 product. If you are using a bus-based 6000 controller, you can use this module to transfer (download) the soft operating system.
- **Test Panel and Program Tester**: You can create your own test panel to run your programs and check the activity of I/O, motion, system status, etc. This can be invaluable during start-ups and when fine tuning machine performance.
- **On-line Context-sensitive Help and Command Reference**: These on-line resources provide help information about Motion Architect, as well as interactive access to the contents of the ***6000 Series Programmer's Guide*** (this document) and the ***6000 Series Software Reference***.
- **Dynamic Link Libraries**: DLL device drivers are provided for bus-based controller customers who wish to create a Windows-based application to interface with the controller.

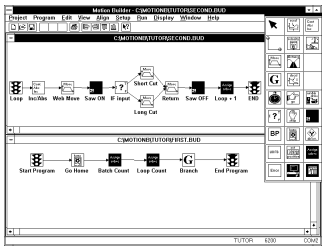
Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Add-On Modules

Add-on modules for Motion Architect are available to aide in other programming and setup tasks. These modules are available from your local Automation Technology Center (ATC) or distributor. For detailed user information, please refer to the respective user guide.

- **Servo Tuner™** (Tuning and Data Gathering Tool): Tune the servo drives and the 6000 servo controller and receive instant data feedback on customizable displays.
- **CompuCAM™**: CAD-to-Motion (CAM) software allows you to translate DXF, HP-GL, and G-Code files into 6000 Series Language motion programs.

Motion Builder™



Motion Builder, a Microsoft Windows-based iconic programming environment, allows expert and novice programmers to easily program Compumotor's 6000 Series products without learning a new programming language or syntax. Use Motion Builder to completely configure the motion controller; program the motion with drag-and-drop visual icons; compile, run and debug the program.

Motion Toolbox™

Motion Toolbox is a library of LabVIEW® virtual instruments (VIs) for Compumotor's 6000 Series controllers. Motion Toolbox allows LabVIEW programmers to develop motion control systems for a wide range of applications including automated test and manufacturing, medical and biotech, metering and dispensing, machine control, and laboratory automation.

Motion Toolbox provides developers with these capabilities:

- Motion control, including velocity, acceleration, deceleration, go, stop, kill, etc.
- Setup, control, and command file transfer
- Counter and timer configuration and control
- Indexer, encoder, and drive configuration
- Home, hardware limit, and soft limit configuration
- Jogging and joystick configuration
- I/O setup and function configuration
- Fast status querying of I/O, limit, home, motor and encoder position, velocity, etc.

DOS Support Software

In addition to Motion Architect, support software written for the DOS environment is available for all 6000 Series products.

Details about these software tools are provided in Chapter 2, *Communication*.

The *6000 DOS Support Disk*, which provides a program for terminal emulation and program editing for serial ("stand-alone") products, is available from your local ATC or distributor.

Bus-based products are shipped with a DOS support disk (see diskette labeled with the product's name) that includes the soft operating system (.OPS file) and programs that demonstrate how to communicate with the 6000 product.

Technical Support

For solutions to your questions about implementing 6000 product software features, first look in this manual. Other aspects of the product (command descriptions, hardware specs, I/O connections, graphical user interfaces, etc.) are discussed in the respective manuals listed above in *Reference Documentation* (see page ii).

If you cannot find the answer in this documentation, contact your local Automation Technology Center (ATC) or distributor for assistance.

If you need to talk to our in-house application engineers, please contact us at the numbers listed on the inside cover of this manual. (The phone numbers are also provided when you issue the `HELP` command to the 6000 controller.)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

CHAPTER ONE

1 Programming Fundamentals

IN THIS CHAPTER

This chapter is a guide to general 6000 programming tasks. It is divided into these main topics:

- Motion Architect programming environment 2
- Command syntax 3
- Creating programs (program development scenario) 8
- Storing programs 12
- Executing programs 14
- Creating and executing a set-up program 14
- Program Security 15
- Controlling execution – programs & command buffer 16
- Restricted commands during motion 18
- Using Variables 18
- Program flow control 23
- Program interrupts 29
- Error handling 30
- Non-volatile memory (stand-alone products) 33
- System performance considerations 33

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Motion Architect Programming Environment

Every 6000 Series controller is shipped with Motion Architect, a Windows-based programming tool designed to simplify your programming efforts. The main features of Motion Architect are briefly described below. For detailed user information, refer to the ***Motion Architect User Guide***.

- **Setup Module**: Provides dialog boxes for you to select basic system setup parameters (I/O definitions, position feedback, etc.) and then automatically generates a fully-commented "setup program."
- **Editor Module**: Create blocks or lines of 6000 controller code, or copy portions of code from previous files. You can save program editor files for later use in BASIC, C, etc., or in the terminal emulator or test panel.
- **Terminal Module**: Communicating directly with the 6000 controller, the terminal emulator allows you to type in and execute controller code and transfer code files to and from the 6000 controller.
- **Panel Module**: You can create your own test panel to run your programs and check the activity of I/O, motion, system status, etc. This can be invaluable during start-ups and when fine tuning machine performance.
- **On-line Help and User Documentation**: Under the Help menu, you will find user information about Motion Architect, as well as interactive access to the contents of the ***6000 Series Programmer's Guide*** (the document you are reading right now) and the ***6000 Series Command Reference***.

Add-on modules for Motion Architect are available to aide in other programming and set-up tasks. These modules are available through your local Automation Technology Center.

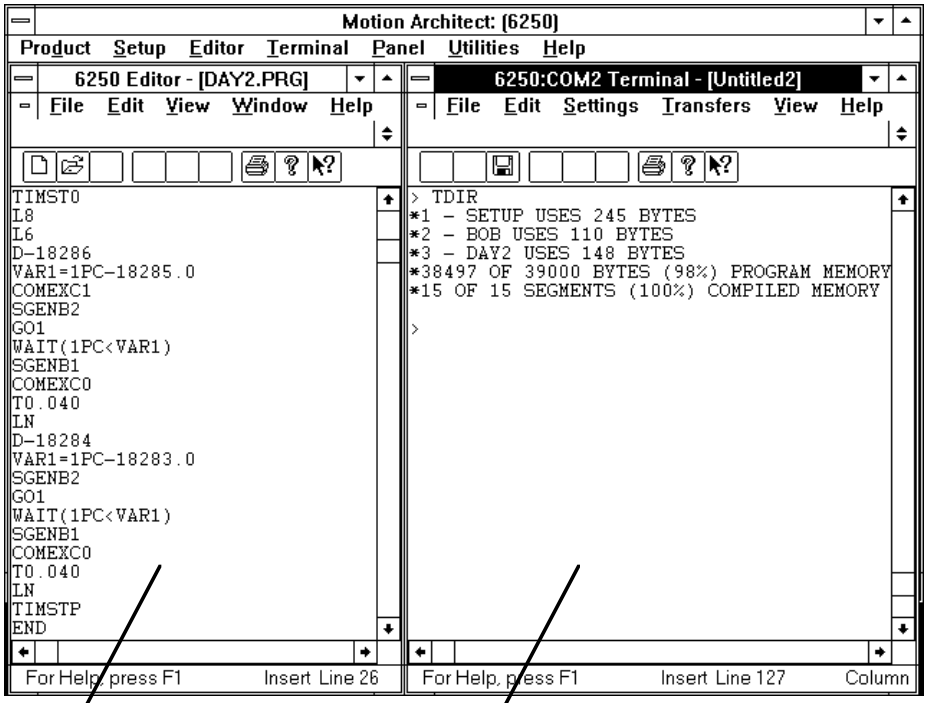
- **Servo Tuner™**: Tune your servo controller and the attached servo drives and receive instant data feedback on customizable displays. For detailed user information, refer to the ***Servo Tuner User Guide***.
- **CompuCAM™**: CompuCAM allows you to import 2D geometry from CAD programs (DXF), plotter files (HP-GL), or NC programs (G-Code), and then translate the geometry into 6000 motion programs. These programs can be further edited in Motion Architect's Program Editor module and dowloaded to the 6000 controller from the Terminal Emulator or Test Panel modules. A typical use of CompuCAM is to automate the process for developing 6000 Series contouring code for an application. For detailed user information, refer to the ***CompuCAM User Guide***.

Side-by-Side Editor and Terminal Windows *(see illustration below)*

This side-by-side technique is demonstrated in the programming scenario on page 8.

Typically, the programming process is an iterative exercise in which you create a program, test it, edit it, test it ... until you are satisfied with the results. To help with this iterative process, we suggest using Motion Architect's Editor and Terminal modules in a side-by-side fashion (open an Editor session and a Terminal session and re-size the windows so that you can see both at the same time). In doing so you can quickly jump back and forth between editing a program (Editor function) and downloading it to the product and checking programming responses and error messages (Terminal functions).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



Program Editor: Create and edit programs, save them, and then download them from the Terminal module.

Terminal Emulator: Communicate directly with the 6000 controller. Download files containing stand-alone commands and/or complete programs or subroutines. Check system responses. Upload programs from the 6000 controller.

Command Syntax

Overview

The 6000 Series language provides high-level constructs as well as basic motion control building blocks. The language comprises simple ASCII mnemonic commands, with each command separated by a command delimiter. Upon receiving a command followed by a command delimiter, the 6000 controller places the command in its internal command queue. Here the command is executed in the order in which it is received. The command may be specified as *immediate* by placing an optional exclamation point (`!`) in front of the command. When a command is specified as an immediate command, it is placed at the front of the command queue, where it is executed immediately.

The command delimiter can be one of three characters, a carriage return, a line-feed, or a colon (`:`). Spaces and tabs within a command are processed as neutral characters. Comments can be specified with the semicolon (`;`) character — all characters following the semicolon until the command delimiter are considered program comments.

There is no case sensitivity with the command language. For instance, the command `TSTAT` is the same as the command `tstat`.

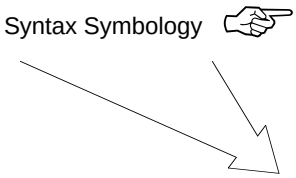
Some commands contain one or more data fields in which you can enter numeric or binary values or text. The `A` command (syntax: `A<r>,<r>,<r>,<r>`) is an example of a command that requires you to enter numeric values (e.g., the `A5,6,7,8` command assigns acceleration values of 5, 6, 7, and 8 units/sec^2 to axes #1, #2, #3, and #4 respectively). The `DRIVE` command (syntax: `DRIVE<b><b><b><b>`) is an example of a command that requires binary values (e.g., the `DRIVE11ØØ` command enables drives #1 and #2 and disables drives #3 and #4). The `STARTP` command (syntax: `STARTP<t>`) is an example of a command that requires text (e.g., the `STARTP powrup` command assigns the program called "`powrup`" as the start-up program).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Description of Syntax Letters and Symbols

The command descriptions provided within the ***6000 Series Software Reference*** use alphabetic letters and ASCII symbols within the **Syntax** description to represent different parameter requirements (see example below).

Syntax Symbology



INEN **Input Enable**

		Product	Rev
Type	Inputs or Program Debug Tools	AT6n00	1.0
Syntax	<!>INEN<d><d><d>...<d>>	AT6n50	1.0
Units	d = Ø, 1, E, or X	610n	1.0
Range	Ø = off, 1 = on, E = enable, X = don't change	615n	1.0
Default	E	620n	1.0
Response	INEN: *INENEEEE_EEEE_EEEE_EEEE_EEEE_EEEE_EEEE	625n	1.0
See Also	[IN], INFEN, INFNC, INLVL, INPLC, INSTW, TIN	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,Ø, 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,Ø, 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 an 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 for more information.
@	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 within the syntax of 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 Ø, 1, X, or x. Examples are assignments such as VARB1=b1ØØØ1, and comparisons such as IF(IN=b1ØØ1X1).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

General Guidelines for Syntax

Topic	Guideline	Examples *
Neutral Characters: • Space (*<sp>*) • Tab (*<tab>*)	Using neutral characters anywhere within a command will not affect the command. (In the examples on the right, a space is represented by *<sp>*, a tab is *<tab>)*, and a carriage return is *<cr>*.)	Set velocity on axis 1 to 10 rps and axis 2 to 25 rps: `V`*<sp>*`1Ø,`*<sp>*`25,,`*<cr>* Add a comment to the command: `V 1Ø, 25,,`*<tab>* `;set accel` *<cr>*
Command Delimiters: • Carriage rtn (*<cr>*) • Line feed (*<lf>*) • Colon (`:`)	All commands must be separated by a command delimiter. A carriage return is the most commonly used delimiter. To use a line in a live terminal emulator session, press ctrl/J. The colon (`:`) delimiter allows you to place multiple commands on one line of code, but only if you add it in the program editor (not during a live terminal emulator session).	Set acceleration on axis 2 to 10 rps^2: `A,1Ø,,`*<cr>* `A,1Ø,,`*<lf>* `A,1Ø,, : V,25,, : D,25ØØØ,, : @GO`*<cr>*
Comment Delimiter (;)	All text between a comment delimiter and a command delimiter is considered *program comments*.	Add a comment to the command: `V1Ø`*<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, 25, 5 and 10 rps, respectively: `V1Ø,25,5,1Ø`*<cr>* Initiate motion on axes 1, 3 and 4: `GO1Ø11`*<cr>* `GO1,Ø,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: `@V1Ø`*<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>*

Case Sensitivity	There is no case sensitivity. Use upper or lower case letters within commands.	Initiate motion on axes 1, 3 and 4: `GO1Ø11`*<cr>* `go1Ø11`*<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).

* Non-visible characters are represented: space = *<sp>*, tab = *<tab>*, carriage return (or enter key) = *<cr>*, line feed = *<lf>*.

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

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

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 command fields.

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

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

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

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

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

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

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 homing 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 homing 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 command syntax) will accept the VARB substitution. Commands with a real or integer command field (<r> or <i> in the command syntax) will accept VAR, READ, DREAD, DREADF, TW or DAT.

Assignment and Comparison Operators

Comparison and assignment operators are used in command arguments for various functions such as variable assignments, conditional branches, wait statements, conditional GOs, etc. Some examples are listed below:

- Assign to numeric variable #6 the value of the encoder position on axis #3 (uses the PE operator): VAR6=3PE
- Wait until inputs #3 & #6 become active (uses the IN operator): WAIT(IN=bxx1xx1)
- Continue until the value of numeric variable #2 is less than 36: UNTIL(VAR2<36)
- IF condition based on if a target zone timeout occurs on axis 2 (uses the AS axis status operator, where status bit #25 is set if a target zone timeout occurs): IF(2AS.25=b1)

The available comparison and assignment operators are listed below. For full descriptions, refer to their respective descriptions in the ***6000 Series Software Reference*** (be sure to refer only to the commands in brackets—e.g., A is the acceleration setup command, but [A] is the acceleration assignment/comparison operator).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

A.....Acceleration
AD .....Deceleration
ANI.....Voltage at the ANI analog inputs (servos with ANI option) *
ANV.....Voltage at the joystick analog channels *
AS .....Axis status *
ASX.....Extended axis status (additional axis status items) *
CA .....Captured ANI analog input voltage *
CNT.....Counter value (steppers only) *
D.....Distance
DAC.....Digital-to-analog converter (output voltage) value (servos only) *
DAT.....Data program number
DPTR.....Data pointer location *
DREAD ....Data from the numeric keypad on the RP240 (stand-alone products only)
DREADF...Data from the function keypad on the RP240 (stand-alone products only)
ER .....Error status *
FB .....Position of current selected feedback sources *
FS .....Following status *
IN .....Input status (input bit patterns provided in Chapter 3, page 107) *
INO.....“Other” input status (joystick inputs, and P-CUT or ENBL input) *
LDT.....Position of the LDT (hydraulic servos only) *
LIM.....Limit status (end-of-travel limits and home limits) *
MOV.....Axis moving status
NMCY.....Current master cycle number *
OUT.....Output status (output bit patterns provided in Chapter 3, page 107) *
PANI.....Position of ANI analog input, at 819 counts/volts unless otherwise scaled (servos) *
PC .....Commanded position (servos only) *
PCA.....Captured ANI input position (servos with ANI option only) *
PCC.....Captured commanded position (servos only) *
PCE.....Captured encoder position *
PCL.....Captured LDT position (hydraulic servos only) *
PCM.....Captured motor position (steppers only) *
PE .....Position of encoder *
PER.....Position error (n/a to OEM-AT6400) *
PM .....Position of motor (steppers only) *
PMAS.....Current master cycle position *
PSHF.....Net position shift since constant following ratio *
PSLV.....Current commanded position of the slave axis *
READ.....Read a numeric value to a numeric variable (VAR)
SEG.....Number of segments available in Compiled Profile memory *
SS .....System status *
TIM.....Timer value *
TW .....Thumbwheel data read
US .....User status *
V.....Velocity (programmed)
VAR.....Numeric variable substitution
VARB.....Binary variable substitution
VEL.....Velocity (commanded by the controller) *
VELA.....Velocity (actual, as measured by a position feedback device) *
VMAS.....Current velocity of the master axis *

* denotes operators that have a correlated status display command.

(e.g., To see a full-text description of each axis status bit accessed with the AS operator, send the TASF command to the 6000 controller.)

See page 232.

Bit Select Operator

The bit select operator (.) makes it easier to base a command argument on the condition of one specific status bit. For example, if you wish to base an IF statement on the condition that a user fault input is activated (error status bit #7 is a binary status bit that is “1” if a user fault occurred and “Ø” if it has not occurred), you could use this command: IF(ER=bxxxxxx1). Using a bit select operator, you could instead use this command: IF(ER.7=b1).

Side Note: You can use a bit select operator to set a particular status bit (e.g., to turn on programmable output #5, you would type the OUT.5-1 command; to enable error-checking bit #4 to check for drive faults, you would type the ERROR.4-1 command). You can also check specific status bits (e.g., to check axis 2's axis status bit #25 to see if a target zone timeout occurred, type the 2TAS.25 command and observe the response).

Binary and Hex Values

When making assignments with or comparisons against binary or hexadecimal values, you must precede the binary value with the letter “b” or “B”, and the hex value with “h” or “H”. Examples: IF(IN=b1xØ1) and IF(IN=h7F). In the binary syntax, an “x’ simply means the status of that bit is ignored. Refer also to *Using Binary Variables* (page 22).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Related Operator Symbols

Command arguments include special operator symbols (e.g., `+`, `/`, `&`, `'`, `>=`, etc.) to perform bitwise, mathematical, relational, logical, and other special functions. These operators are described in detail, along with programming examples, at the beginning of the *Command Descriptions* section of the ***6000 Series Software Reference***.

Programmable Inputs and Outputs Bit Patterns

I/O pin outs, specifications, and circuit drawings are provided in each 6000 Series product's installation guide.

The total number of inputs and outputs (I/O) varies from one 6000 Series product to another. Consequently, bit patterns for the programmable I/O also vary by product. For example, the AT6400's **TRG-A** trigger input is represented by programmable input bit #25, but the 6104's **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 offering and bit patterns, refer to Chapter 3 (page 107).

Creating Programs

A *program* is a series of commands. These commands are executed in the order in which they are programmed. Immediate commands (commands that begin with an exclamation point [`!`]) cannot be stored in a program. Only buffered commands may be used in a program.

Debugging Programs: Refer to page 231 for methods to isolate and resolve programming problems.

A *subroutine* is defined the same as a program, but it is executed with an unconditional branch command, such as `GOSUB`, `GOTO`, or `JUMP`, from another program (see page 23 for details about unconditional branching). Subroutines can be nested up to 16 levels deep. NOTE: The 6000 family does not support recursive calling of subroutines.

Another kind of program is a *compiled profile*. 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). For more information on contours, refer to *Contouring* in Chapter 5, page 153. For more information on compiled individual axis profiles, refer to *Compiled Motion Profiling* in Chapter 5, page 163.

Program Development Scenario

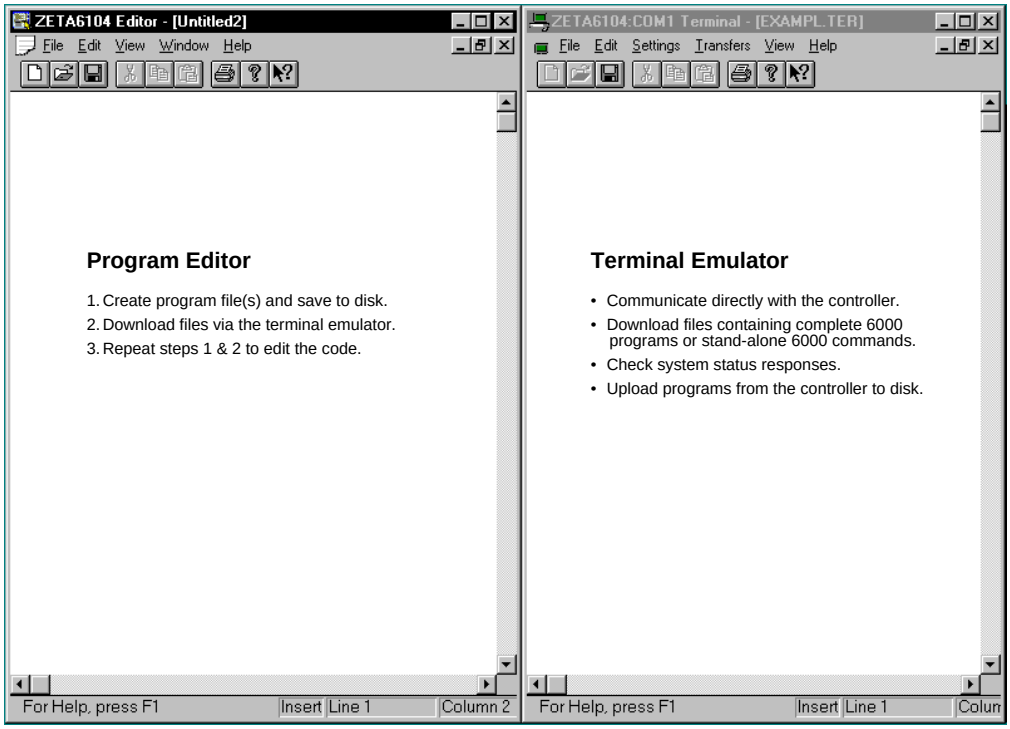
To best understand the process of developing 6000 programs, we invite you to follow along on a program development scenario. The scenario covers these programming tasks:

1. Set up the programming environment (using Motion Architect's Editor and Terminal).
2. Create a simple motion program in the Editor.
3. Download the program, via the Terminal, to the controller.
4. Execute (run) the program from the Terminal.
5. Edit the program, redownload it, and run it again.

1. Set up Motion Architect.

Typically, the programming process is an iterative exercise in which you create a program, test it, edit it, test it ... until you are satisfied with the results. To help with this iterative process, we suggest using Motion Architect's Editor and Terminal modules in a side-by-side fashion (open an Editor session and a Terminal session and re-size the windows so that you can see both at the same time). In doing so you can quickly jump back and forth between editing a program (Editor function) and downloading it to the product and checking programming responses and error messages (Terminal functions).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



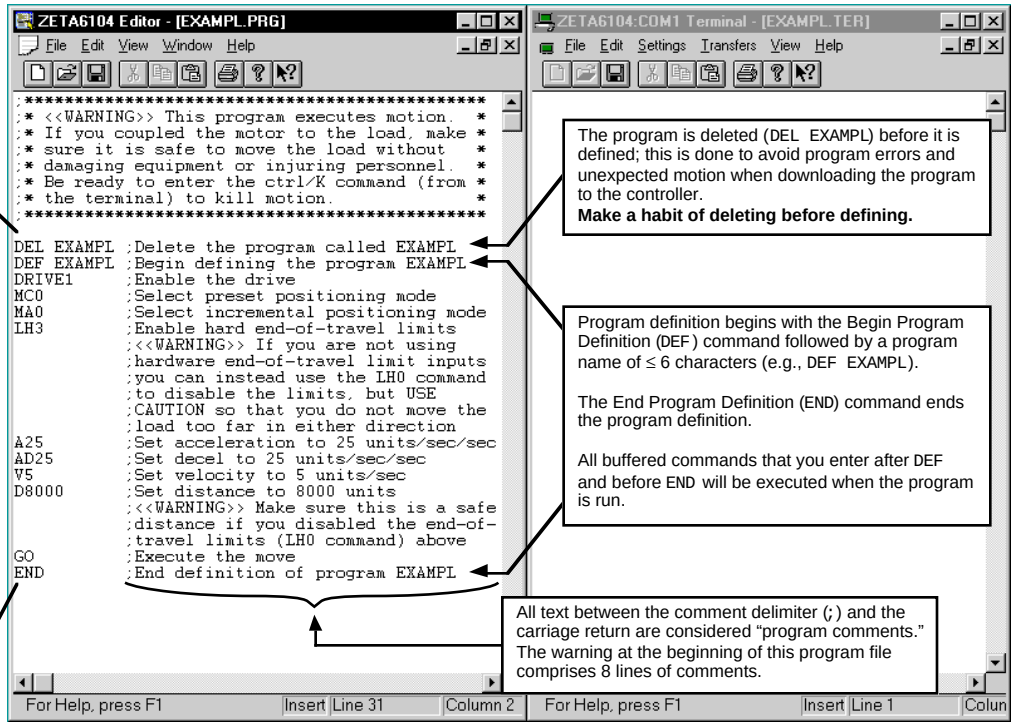
2. Define a program.

Type in the commands as shown in the Editor window below. The Editor window also shows program comments to help you understand the purpose of each command and the implications of executing motion. Notice that the comments are placed after the comment delimiter (`;`).

NOTE: This is a programming example for single-axis motion. When programming multiple axes, you would use the additional command fields. For example, the command for setting the acceleration on axes one and two to 12 units/sec^2 and axes three and four to 25 units/sec^2 would be `A12,12,25,25`; and setting axes one and two to preset positioning mode and axes three and four to continuous positioning mode requires the `MCØØ11` command.

Use these commands:

```
DEL EXAMPL
DEF EXAMPL
DRIVE1
MC0
MA0
LH3 (or LH0)
A25
AD25
V5
D8000
GO
END
```

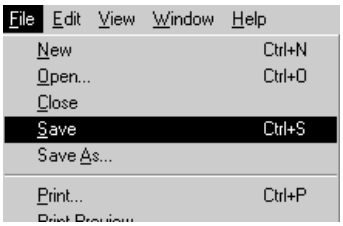


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

3. Download the program.

FIRST: Before you can download the program, you must save it to your hard drive (select File/Save in the Editor). The file is saved with a ".PRG" extension. In the Save dialog box, name the file "EXAMPL.PRG".

Save the program first (Editor).

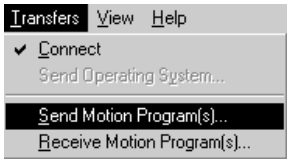


Follow these steps to download the program file to the controller:

1. Power up the controller. Notice the power-up message (like the one shown below) display in your terminal emulator window.

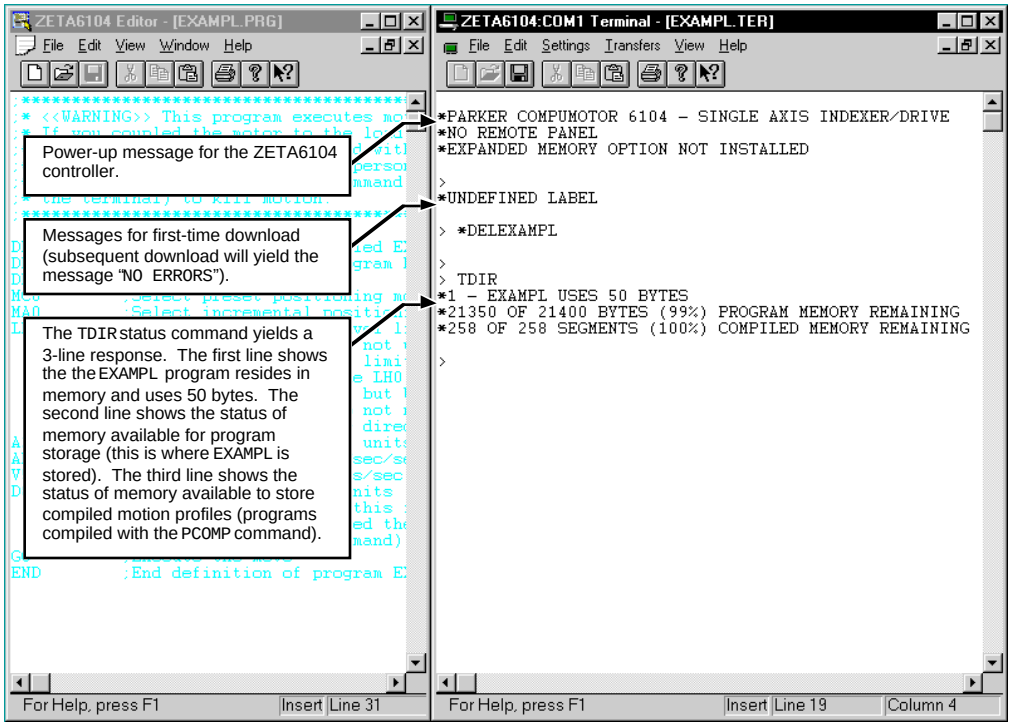
2. From the Transfers menu in the Terminal window, select Send Motion Program(s). From the dialog box, select the EXAMPL.PRG file and click OK. The first time you download the file, you will see two messages display: "`*UNDEFINED LABEL`" and "`DELEXAMPL`" (on subsequent downloads of the same file, you would instead see the message "`NO ERRORS`").

Download the program (Terminal).



> **TIP**: If the controller is executing a program when you try to download a program file, the program(s) from the program file will not download and the contents of the program(s) will be displayed to the terminal emulator window. To prevent this error, "kill" program execution (and motion) before downloading the program file — use the `!K` command or the ctrl/K command (ctrl/K is an easier keystroke combination).

3. To verify that the `EXAMPL` program resides in the controller's memory, type the `TDIR` command and press the carriage return or enter key.

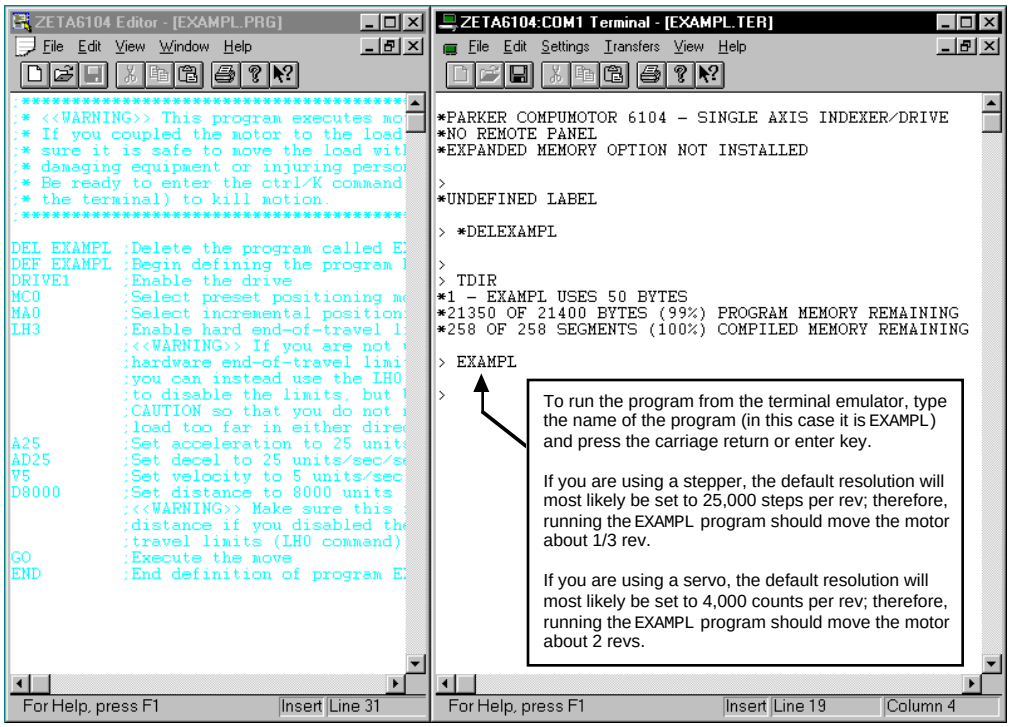


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

4. Run the program.

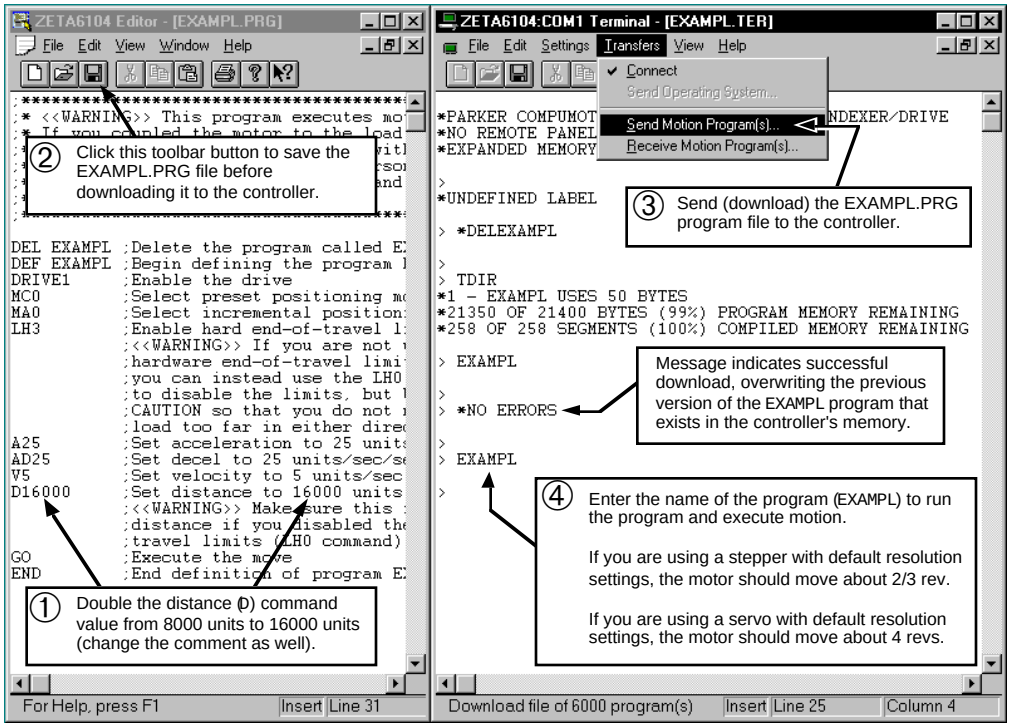
WARNING: Executing this program will cause motion. If you coupled the motor(s) to the load, make sure it is safe to move the load without damaging equipment or injuring personnel.

Other methods of executing programs are listed in *Executing Programs* (page 14).



5. Edit the program.

To demonstrate the iterative nature of programming, let's change distance of the move in the EXAMPL program, redownload the EXAMPL.PRG file, and run the EXAMPL program.



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Storing Programs

Programs and compiled profiles are stored in the controller's memory (non-volatile memory for stand-alone products and volatile memory for bus-based products). Information on controlling memory allocation is provided below (see *Memory Allocation*).

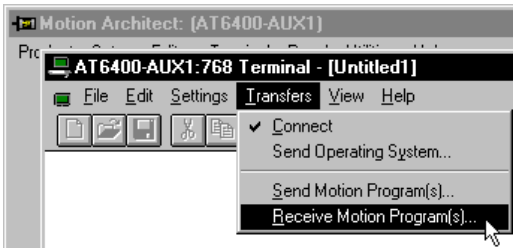
Storing Programs in Stand-Alone Products

If you are using a stand-alone (serial interface) product, programs and compiled profiles are automatically stored in non-volatile memory (battery-backed RAM).

More information on other items that are stored in non-volatile memory is provided below.

Storing Programs in Bus-Based Products

If you are using a bus-based product, programs and compiled profiles are stored in volatile RAM memory (*not battery-backed*). Therefore, you should **backup your motion programs** to your computer's hard disk or floppy disk to ensure their safety. This is easily done with the Receive Motion Program function of Motion Architect's Terminal Emulator module (see diagram at right).



In general, your programs may already be stored on your computer, since most programs are created with Motion Architect's Editor or with the 6000 DOS support software package (see *Program Development Scenario* starting on page 8).

Application set-up parameters such as drive setup, feedback setup, I/O configuration, etc., should be placed in a set-up program that is called/downloaded and executed before performing any other controller functions (see *Creating and Executing a Set-Up Program* on page 14).

Memory Allocation

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* starting on page 8.

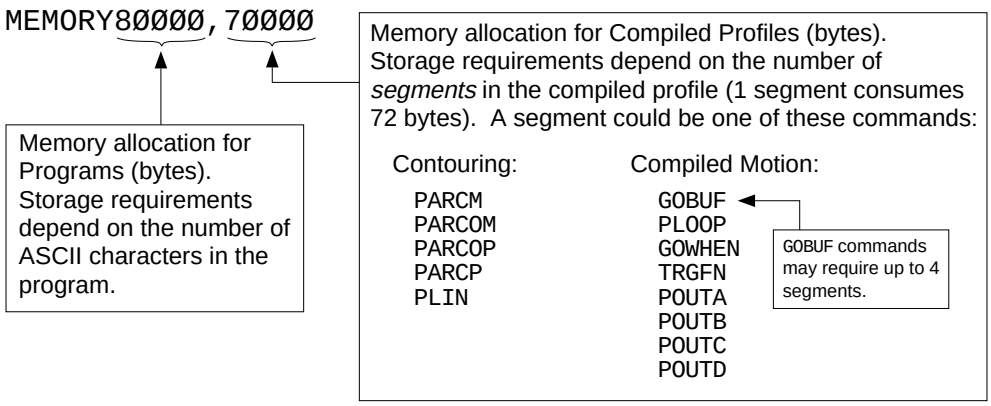
Compiled Profiles are defined like programs (using the DEF and END commands), but are compiled with the PCOMP command and executed with the PRUN command. A compiled profile could be a multi-axis *contour* (a series of arcs and lines), an *individual axis profile* (a series of GOBUF commands), or a *compound 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 Chapter 5, page 153. For more information on compiled profiles for individual axes, refer to *Compiled Motion Profiling* in Chapter 5, page 163.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

MEMORY command syntax (example)



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 (see *Variable Arrays* in Chapter 3, page 120, for details).

CAUTION

Using a memory allocation command (e.g., MEMORY39000,1000) will erase all existing programs and compiled profile segments. However, issuing the MEMORY command without parameters (i.e., type MEMORY *<cr>* 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 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.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Executing Programs (options)

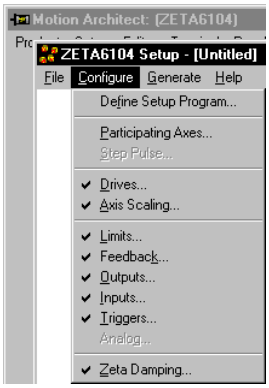
Following is a list of the primary options for executing programs stored in your controller:

Method	Description	See Also
Execute from a terminal emulator	Type in the name of the program and press enter.	page 11
Execute as a subroutine from a "main" program	Use an branch (`GOTO`, `GOSUB`, or `JUMP`) from the main program to execute another stored program.	page 23
Execute automatically when the controller is powered up	Stand-alone products only. Assign a specific program as a startup program with the `STARTP` command. When you `RESET` or cycle power to the controller, the startup program is automatically executed.	page 14
Execute a specific program with BCD weighted inputs	Define programmable inputs to function as BCD select inputs, each with a BCD weight. A specific program (identified by its number) is executed based on the combination of active BCD inputs. Related commands: `INFNCi-B`, `INSELP`, `INFEN`.	page 110
Execute a specific program with a dedicated input	Define a programmable input to execute a specific program (by number). Related commands: `INFNCi-iP`, `INSELP`, `INFEN`.	page 115
"Call" from a high-level program	Using a programming language such as BASIC or C, write a program that enables the computer to monitor processes and orchestrate motion and I/O by executing stored programs (or individual commands) in the controller.	page 143
Execute from an RP240 (remote operator interface)	Execute a stored program from the `RUN` menu in the RP240's standard menu system.	page 134
Execute from your own custom Windows program	Bus-based products only. Use a programming language such as Visual Basic and the DLLs provided on your Motion Architect disk to create your own windows application to control the 6000 product.	page 51
Execute from Motion Architect's Panel module	Use Motion Architect's Panel module to create a test/operator panel screen to execute programs and monitor the controller's status (I/O, motion, axis, system, Following, etc.)	*Motion Architect User Guide*

Creating and Executing a Set-up Program

In most applications, you will benefit by having a *set-up*, or *configuration*, program that is executed before performing any other controller functions. The set-up program contains various set-up parameters specific to the general operation of your controller. Examples of these parameters include scaling factors, I/O definitions, feedback device configuration, homing operations, end-of-travel limits, drive configuration, program execution modes, etc. (more detail on these features is provided in Chapter 3, *Basic Operation Setup*, page 77).

Use Motion Architect's Setup module to help you create the basic configuration program. By simply responding to a series of dialog boxes, a program is created with a specific name (as if you created it in the usual process with the `DEF` and `END` commands, as demonstrated on page 9). You can further edit this program in Motion Architect's Editor module if you wish. How you execute the set-up program depends on which product form factor you are using: stand-alone or bus-based.



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Set-up Program Execution for Stand-Alone Controllers

If you created the set-up program in Motion Architect's Editor, you need to download it to the 6000 controller's non-volatile memory via the Terminal Emulator module (see Send Motion Program under the Transfers menu). If you created the set-up program in the terminal emulator, as in the example below, it is already stored to non-volatile memory.

Now that the set-up program is available, you can cause it to be executed automatically after the 6000 controller is powered-up or reset. To do this, you must assign it as the power-up start program with the STARTP command (see fourth line in example below).

Example

```
DEF setup               ; Define program setup
TREV                    ; Report software revision
END                     ; End of program setup
STARTP setup            ; Define program pwrup as the power-up program
RESET                   ; Reset the controller
;   After reset, you should see a message like this:
;                   *PARKER COMPUMOTOR 6201 MOTION CONTROLLER
;                   *NO REMOTE PANEL
```

If the program that is identified as the STARTP program is deleted by the DEL command, the STARTP is automatically cleared. If you wish to prevent the assigned STARTP program from being executed, without having to delete the program, issue the STARTP CLR command.

Set-up Program Execution for Bus-Based Controllers

In most cases you will require the parameters in the setup program to be executed as soon as possible so that subsequent parameters are based on the setup program. This can be done using Motion Architect. A set up program can be defined (in Motion Architect's *Setup* Module), saved, and then downloaded in the *Terminal Module* (see *Send Motion Program* under the *Transfers* Menu). Once the setup program has been stored in the controller, it may be run by issuing the name of the setup program.

An alternative method would be to not store the setup parameters in a setup program, but have them execute upon downloading to the controller. This can be done be defining the setup parameters in the *Setup* Module of Motion Architect, but not specifying a setup program. This will remove the DEF and END statements from the setup file, which you will download the same way in Motion Architect's *Terminal* Module. Because the statements execute upon downloading, there is no need to issue a program name.

Program Security

Issuing the INFNCi-Q command enables the *Program Security* feature and assigns the *Program Access* function to the specified programmable input. The "i" represents the number of the programmable input to which you wish to assign the function.

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.

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.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Controlling Execution of Programs and the Command Buffer

The 6000 controller command buffer is capable of storing 2000 characters waiting to be processed. (*This is separate from the memory allocated for program storage – see* Memory Allocation, *page 12.*) Three commands, COMEXC, COMEXK, and COMEXP, affect command execution. Three additional commands, COMEXL, COMEXR, and COMEXS, affect the execution of programs and the command buffer.

COMEXC (Continuous Command Execution)

The COMEXC command enables the Continuous Command Execution Mode. This mode allows the program to continue to the next command before motion is complete. This is useful for:

- Monitoring other processes while motion is occurring
- Performing calculations in advance of motion completion
- Pre-emptive GOs — executing a new profile with new attributes (distance, accel/decel, velocity, positioning mode, and Following ratio) before motion is complete: 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. For a detailed description, refer to *On-The-Fly Motion* on page 178.
- Pre-process the next move while the current move is in progress (see CAUTION note). 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 *Target Zone Mode* on page 105 for details).

In the programming example below, by enabling the continuous command execution mode (COMEXC1), the controller is able to turn on output #3 after the encoder moves 4000 units of its 125000-unit move. Normally, with COMEXC disabled (COMEXCØ), command processing would be temporarily stopped at the GO1 command until motion is complete.

Programming Example (portion of program only)

```
COMEXC1          ;Enable continuous command execution mode
D125000          ;Set distance
V2               ;Set velocity
A10              ;Set acceleration
GO1              ;Initiate motion on axis 1
WAIT(1PE>4000)   ;Wait for the encoder position to exceed 4000
OUTXX1           ;Turn on programmable output #3
WAIT(AS.1=b0)    ;Wait for motion to complete on axis 1 (AS bit #1 = zero)
OUTXX0           ;Turn off programmable output #3
```

COMEXK (Continue Command Execution on Kill)

This feature is applicable only to bus-based products. The COMEXK 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.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

COMEXL (Save Command Buffer on Limit)

The COMEXL command enables saving the command buffer and maintaining program execution when a hardware or software end-of-travel limit is encountered. For more information on end-of-travel limits, refer to page 90.

COMEXP (Pause Command Execution Until In Position Signal)

This feature is applicable only to stepper products, excluding the OEM-AT6400, 6104 and 6201. The COMEXP command enables waiting for the in-position signal (**DRIVE** connector pin 4). While enabled, the next command will not be processed until the in-position signal becomes active. This only affects the command processing of motion commands.

COMEXR (Effect of Pause/Continue Input)

The COMEXR command affects whether a pause input (i.e., a general-purpose input configured as a pause/continue input with the INFNCi-E command) will pause only program execution or both program execution and motion.

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

COMEXR1: Upon receiving a pause input, both motion and program execution will be paused; the motion stop function is used to halt motion. *After motion has come to a stop (not during deceleration)*, you can release the pause input or issue a !C command to resume motion and program execution.

Other Ways to Pause

- Issue the PS command before entering a series of buffered commands (to cause motion, activate outputs, etc.), then issue the !C command to execute the commands.
- While program execution is in progress, issuing the !PS command stops program execution, but any move currently in progress will be completed. Resume program execution with the !C command.

COMEXS (Save Command Buffer on Stop)

The COMEXS command affects saving the command buffer and maintaining program execution upon receiving a stop input (a general-purpose input configured with the INFNCi-D command) or a stop command (!S or !S111).

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

COMEXS1: Upon receiving a stop input or stop command, motion will decelerate at the preset AD/ADA value, program execution will pause, and all commands following the command currently being executed will remain in the command buffer.

Resuming program execution (*only after motion has come to a stop*):

- 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/continue input (a general-purpose input configured with the INFNCi-E command—see COMEXR discussion above).
- If you are resuming after a stop input or a !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 where the !S command was received by the processor.

COMEXS2: Upon receiving a stop input or stop command, motion will decelerate at the preset AD value, every command in the command 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.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Restricted Commands During Motion

When motion is in progress, some commands cannot have their parameters changed until motion is complete (see table below).

For the commands identified in the table, if the continuous command execution mode in enabled (COMEXC1) and you try to enter new command parameters, you will receive the error response MOTION IN PROGRESS. If the continuous command execution mode in disabled (COMEXCØ), which is the default setting, you will receive the response MOTION IN PROGRESS only if you precede the command with the immediate (!) modifier (e.g., !V2Ø); if you enter a command without the immediate modifier (e.g., V2Ø), you will not receive an error response and the new parameter will be ignored and the old parameter will remain in effect.

All of the commands in the table below, except for INDAX and SCALE, are axis-dependent. That is, if one axis is moving you can change the parameters on the other axes, provided they are not in motion.

Command	Description
ANIPOL	ANI input Polarity
CMDDIR	Commanded Direction Polarity
DRES	Drive Resolution
DRIVE	Drive Shutdown
ENC	Encoder/Motor Step Mode
ENCPOL	Encoder Polarity
ERES	Encoder Resolution
EPMV	Position Maintenance Max Velocity
FOLEN	Following Mode Enable
FR	Feedrate Enable
GOL	Initiate Linear Interpolated Motion
HOM	Go Home
HOMA	Home Acceleration
HOMAA	Average Home Acceleration
HOMAD	Home Deceleration
HOMADA	Average Home Deceleration
HOMV	Home Velocity
HOMVF	Home Final Velocity
INDAX	Participating Axes *
JOG	Jog Mode Enable
JOGA	Jog Acceleration
JOGAA	Average Jog Acceleration
JOGAD	Jog Deceleration
JOGADA	Average Jog Deceleration
JOGVH	Jog Velocity High
JOGVL	Jog Velocity Low

Command	Description
JOY	Joystick Mode Enable
JOYA	Joystick Acceleration
JOYAA	Average Joystick Acceleration
JOYAD	Joystick Deceleration
JOYADA	Average Joystick Deceleration
JOYVH	Joystick Velocity High
JOYVL	Joystick Velocity Low
LDTPOL	LDT Polarity
LDTRES	LDT Resolution
LHAD	Hard Limit Deceleration
LHADA	Average Hard Limit Deceleration
LSAD	Soft Limit Deceleration
LSADA	Average Soft Limit Deceleration
PSET	Establish Absolute Position
SCALE	Enable/Disable Scale Factors *
SCLA	Acceleration Scale Factor
SCLD	Distance Scale Factor
SCLV	Velocity Scale Factor
SSV	Start/Stop Velocity

* If any axis is in motion, you will cause an error if you attempt to change this command's parameters.

Variables

6000 Series controllers have three types of variables (numeric, binary, and string). Each type of variable is designated with a different command: VAR (numeric variable), VARB (binary variable), and VARS (string variable). The quantity available for each variable type differs by product:

Variable Type	AT6n00	AT6n00-M (with expanded memory)	All Other 6000 Controllers (regardless of -M option)
Numeric (VAR)	100	200	150
Binary (VARB)	100	100	25
String (VARS)	100	100	25

NOTE: Variables do not share the same memory (e.g., VAR1, VARB1, and VARS1 can all exist at the same time and operate separately).

Numeric variables are used to store numeric values with a range of -999,999,999.ØØØØØØØØ to 999,999,999.99999999. Mathematical, trigonometric, and boolean operations are performed using numeric variables. You can also use numeric variables to store ("teach") variable data in variable arrays (called *data programs*) and later use the stored data as a source for motion program parameters (see *Variable Arrays* on page 120 for details).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Stand-Alone Products:
- *All 3 types of variables are automatically storied in non-volatile memory.*
- *6270: Numeric and string variables may be displayed with the RP240 (see page 133).*

Binary variables can be used to store 32-bit binary or hexadecimal values. Binary variables can also store I/O, system, axis, or error status (e.g., the VARB2=IN.12 command assigns input bit 12 to binary variable 2). Bitwise operations are performed using binary variables.

String variables are used to store message strings of 20 characters or less. These message strings can be predefined error messages, user messages, etc. The programming example in the *Command Value Substitutions* (page 6) demonstrates the use of a string variable.

Converting Between Binary and Numeric Variables

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. Any *don't care*s (X) in a binary value is 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

Example	Description/Response
VAR1=-5	Set numeric variable value = -5
VARB1=VCVT(VAR1)	Convert the numeric value to a binary value
VARB1	*VARB1=1101_1111_1111_1111_1111_1111_1111_1111

Binary to Numeric

Example	Description/Response
VARB1=b0010_0110_0000_0000_0000_0000_0000_0000	Set binary variable = +100.0
VAR1=VCVT(VARB1)	Convert binary value to numeric
VAR1	*VAR1=+100.0

Using Numeric Variables

Some Numeric Variable Operations Reduce Precision

The following operations reduce the precision of the return value: Division and Trigonometric functions yield 5 decimal places; Square Root yields 3 decimal places; and Inverse Trigonometric functions yield 2 decimal places.

Mathematical Operations

The following examples demonstrate how to perform math operations with numeric variables. Operator precedence occurs from left to right (e.g., VAR1=1+1+1*3 sets VAR1 to 9, not 5).

Addition (+)

Example	Response
VAR1=5+5+5+5+5+5+5	
VAR1	*VAR1=35.0
VAR23=1000.565	
VAR11=VAR1+VAR23	
VAR11	*VAR11=+1035.565
VAR1=VAR1+5	
VAR1	*VAR1=+40.0

Subtraction (-)

Example	Response
VAR3=20-10	
VAR20=15.5	
VAR3=VAR3-VAR20	
VAR3	*VAR3=-5.5

Multiplication (*)

Example	Response
VAR3=10	
VAR3=VAR3*20	
VAR3	*VAR3=+200.0

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Division (/)

Example	Response
VAR3=10	
VAR20=15.5	
VAR20	*+15.5
VAR3=VAR3/VAR20	
VAR3	*+0.64516
VAR30=75	
VAR30	*+75.0
VAR19=VAR30/VAR3	
VAR19	*+116.25023

Square Root

Example	Response
VAR3=75	
VAR20=25	
VAR3=SQRT(VAR3)	
VAR3	*+8.660
VAR20=SQRT(VAR20)+SQRT(9)	
VAR20	*+8.0

Trigonometric Operations

The following examples demonstrate how to perform trigonometric operations with numeric variables.

Sine

Example	Response
RADIAN0	
VAR1=SIN(0)	
VAR1	*VAR1=+0.0
VAR1=SIN(30)	
VAR1	*VAR1=+0.5
VAR1=SIN(45)	
VAR1	*VAR1=+0.70711
VAR1=SIN(60)	
VAR1	*VAR1=+0.86603
VAR1=SIN(90)	
VAR1	*VAR1=+1.0
RADIAN1	
VAR1=SIN(0)	
VAR1	*VAR1=+0.0
VAR1=SIN(PI/6)	
VAR1	*VAR1=+0.5
VAR1=SIN(PI/4)	
VAR1	*VAR1=+0.70711
VAR1=SIN(PI/3)	
VAR1	*VAR1=+0.86603
VAR1=SIN(PI/2)	
VAR1	*VAR1=+1.0

Cosine

Example	Response
RADIAN0	
VAR1=COS(0)	
VAR1	*VAR1=+1.0
VAR1=COS(30)	
VAR1	*VAR1=+0.86603
VAR1=COS(45)	
VAR1	*VAR1=+0.70711
VAR1=COS(60)	
VAR1	*VAR1=+0.5
VAR1=COS(90)	
VAR1	*VAR1=+0.0
RADIAN1	
VAR1=COS(0)	
VAR1	*VAR1=+1.0
VAR1=COS(PI/6)	
VAR1	*VAR1=+0.86603
VAR1=COS(PI/4)	
VAR1	*VAR1=+0.70711
VAR1=COS(PI/3)	
VAR1	*VAR1=+0.5
VAR1=COS(PI/2)	
VAR1	*VAR1=+0.0

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Tangent

```
Example                  Response
RADIAN0
VAR1=TAN(0)
VAR1                     *VAR1=+0.0
VAR1=TAN(30)
VAR1                     *VAR1=+0.57735
VAR1=TAN(45)
VAR1                     *VAR1=+1.0
VAR1=TAN(60)
VAR1                     *VAR1=+1.73205
RADIAN1
VAR1=TAN(0)
VAR1                     *VAR1=+0.0
VAR1=TAN(PI/6)
VAR1                     *VAR1=+0.57735
VAR1=TAN(PI/4)
VAR1                     *VAR1=+1.0
VAR1=TAN(PI/3)
VAR1                     *VAR1=+1.73205
```

Inverse Tangent (Arc Tangent)

```
Example                  Response
RADIAN0
VAR1=SQRT(2)
VAR1=ATAN(VAR1/2)
VAR1                     *VAR1=+35.26
VAR1=ATAN(.57735)
VAR1                     *VAR1=+30.0
```

Boolean Operations

6000 Series products have the ability to perform Boolean operations with numeric variables. The following examples illustrate this capability. Refer to the ***6000 Series Software Reference*** for more information on each operator (&, |, ^, and ~).

Boolean And (&)

```
Example                  Response
VAR1=5
VAR2=-1
VAR3=VAR1 & VAR2
VAR3                     *VAR3=+0.0
```

Boolean Or (|)

```
Example                  Response
VAR1=5
VAR2=-1
VAR3=VAR1 | VAR2
VAR3                     *VAR3=+1.0
```

Boolean Exclusive Or (^)

```
Example                  Response
VAR1=5
VAR2=-1
VAR3=VAR1 ^ VAR2
VAR3                     *VAR3=+1.0
```

Boolean Not (~)

```
Example                  Response
VAR1=5
VAR3=~(VAR1)
VAR3                     *VAR3=+0.0
VAR1=-1
VAR3=~(VAR1)
VAR3                     *VAR3=+1.0
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Using Binary Variables

The following examples illustrate the 6000 Series product's ability to perform bitwise functions with binary variables.

Storing binary values. The 6000 Series Language allows you to store binary numbers in the binary variables (VARB) command. The binary variables start at the left with the least significant bit, and increase to the right. For example, to set bit 1, 5, and 7 you would issue the command `VARB1=b1xxx1x1`. Notice that the letter `b` is required. When assigning a binary value to a binary variable, only the bits specified are affected—all unspecified bits are left in their current state.

```
Example                 Response
VARB1=b1101XX1          *VARB1=1101_XX1X_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX
```

Storing hexadecimal values. Hexadecimal values can also be stored in binary variables (VARB). The hexadecimal value must be specified the same as the binary value—left is least significant byte, right is most significant. For example, to set bit 1, 5, and 7 you would issue the command `VARB1=h15`. Notice that the letter `h` is required. **NOTE**: When assigning a hexadecimal value to a binary variable, all unspecified bits are set to zero.

```
Example                 Response
VARB1=h7FAD
VARB1                   *VARB1=1110_1111_0101_1011_0000_0000_0000_0000
```

Bitwise And (&)

```
Example                 Response
VARB1=b1101
VARB1                   *VARB1=1101_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX
VARB1=VARB1 & bXXX1 1101
VARB1                   *VARB1=XX01_XX0X_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX
VARB1=h0032 FDA1 & h1234 43E9
VARB1                   *VARB1=0000_0000_1100_0000_0010_1000_0101_1000
```

Bitwise Or (|)

```
Example                 Response
VARB1=h32FD
VARB1                   *VARB1=1100_0100_1111_1011_0000_0000_0000_0000
VARB1=VARB1 | bXXX1 1101
VARB1                   *VARB1=11X1_1101_1111_1X11_XXXX_XXXX_XXXX_XXXX
VARB1=h0032 FDA1 | h1234 43E9
VARB1                   *VARB1=1000_0100_1100_0110_1111_1111_0111_1001
```

Bitwise Exclusive Or (^)

```
Example                 Response
VARB1=h32FD ^ bXXX1 1101
VARB1                   *VARB1=XXX1_1001_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX
VARB1=h0032 FDA1 ^ h1234 43E9
VARB1                   *VARB1=1000_0100_0000_0110_1101_0111_0010_0001
```

Bitwise Not (~)

```
Example                 Response
VARB1=~(h32FD)
VARB1                   *VARB1=0011_1011_0000_0100_1111_1111_1111_1111
VARB1=~(b1010 XX11 0101)
VARB1                   *VARB1=0101_XX00_1010_XXXX_XXXX_XXXX_XXXX_XXXX
```

Shift Left to Right (>>)

```
Example                 Response
VARB1=h32FD >> h4
VARB1                   *VARB1=0000_1100_0100_1111_1011_0000_0000_0000
VARB1=b1010 XX11 0101 >> b11
VARB1                   *VARB1=0001_010X_X110_101X_XXXX_XXXX_XXXX_XXXX
```

Shift Right to Left (<<)

```
Example                 Response
VARB1=h32FD << h4
VARB1                   *VARB1=0100_1111_1011_0000_0000_0000_0000_0000
VARB1=b1010 XX11 0101 << b11
VARB1                   *VARB1=0XX1_1010_1XXX_XXXX_XXXX_XXXX_XXXX_X000
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Program Flow Control

Program flow refers to the order in which commands will be executed, and whether they will be executed at all. In general, commands are executed in the order in which they are received. However, certain commands can redirect the order in which commands will be processed.

You can affect program flow with:

- Unconditional Loops and Branches
- Conditional Loops and Branches

Unconditional Looping and Branching

Unconditional Looping

The Loop (L) command is an unconditional looping command. You may use this command to repeat the execution of a group of commands for a predetermined number of iterations. You can nest Loop commands up to 16 levels deep. The code sample (portion of a program) below demonstrates a loop of 5 iterations.

```
MA0       ; Sets unit to Incremental mode
A50       ; Sets acceleration to 50
V5        ; Sets velocity to 5
L5        ; Loops 5 times
D2000     ; Sets distance to 2,000
GO1       ; Executes the move (Go)
T2        ; Delays 2 seconds after the move
LN        ; Ends loop
```

Unconditional Branching

There are three ways to branch unconditionally:

GOTO: The GOTO command transfers control from the current program being processed to the program name or label stated in the GOTO command.

GOSUB: The GOSUB command branches to the program name or label stated in the GOSUB command; however, the GOSUB command returns control to the program where the branch occurred (resumes at the next command line after the GOSUB).

JUMP: The JUMP command branches to the program name or label stated in the JUMP command. 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 branch command will be ignored and processing will continue with the next line in the program.

The 6000 family does not support recursive calling of subroutines.

Using labels: Labels, defined with the $ command, provide a method of branching to specific locations within the same program. Labels can only be defined within a program and executed with a GOTO, GOSUB, or JUMP command from within the same program (*see Example B below*).

NOTE

Be careful about performing a GOTO within a loop or branch statement area (i.e., between L&LN, between IF & NIF, between REPEAT & UNTIL, or between WHILE & NWHILE). Branching to a different location within the same program will cause the next L, IF, REPEAT, or WHILE statement encountered to be nested within the previous L, IF, REPEAT, or WHILE statement area, unless an LN, NIF, UNTIL, or NWHILE command has already been encountered.

** To avoid this nesting situation, use the JUMP command instead of the GOTO command.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Example A DESCRIPTION: The program `cut1` is executed until it gets to the command `GOSUB prompt`. From there it branches unconditionally to the subroutine (actually a program) called `prompt`. The subroutine `prompt` queries the operator for the number of parts to process. After the part number is entered (e.g., operator enters the `!'12` command to process 12 parts), the rest of the `prompt` subroutine is executed and control goes back to the `cut1` program and resumes program execution with the next command after the `GOSUB`, which is `MAØØ`.

```
DEL cut1                ; Delete a program before defining it
DEF cut1                ; Begin definition of program cut1
HOM11                   ; Send axes 1 and 2 to the home position
WAIT(1AS=b0XXX1 AND 2AS=b0XXX1)   ; Wait for axes 1 and 2 to come
                        ; to a halt at home
GOSUB prompt            ; Go to subroutine program called prompt
MA00                    ; Place axes 1 and 2 in the incremental mode
A10,30                  ; Set acceleration: axis 1 = 10, axis 2 = 30
AD5,12                  ; Set deceleration: axis 1 = 5, axis 2 = 12
V5,8                    ; Set velocity: axis 1 = 5, axis 2 = 8
D16000,100000           ; Set distance: axis 1 = 16,000; axis 2 = 100,000
OUT.6-1                 ; Turn on output number 6
T5                      ; Wait for 5 seconds
L(VAR2)                 ; Begin loop ( the number of loops = value of VAR2)
  GO11                  ; Initiate moves on axes 1 and 2
  T3                    ; Wait for 3 seconds
LN                      ; End loop
OUT.6-0                 ; Turn off output number 6
END                     ; End definition of program cut1

DEF prompt              ; Begin definition of program prompt
VARS1="Enter part count >"  ; Place message in string variable #1
VAR2=READ1              ; Prompt operator with string variable #1,
                        ; and read data into numeric variable #2
                        ; NOTE:  Type !' before the part count number.
END                     ; End definition of program prompt
```

Example B This example demonstrates the use of labels ($).

```
DEL pick                ; Delete a program before defining it
DEF pick                ; Begin definition of program pick
GO1100                  ; Initiate motion and axes 1 and 2
IF(VAR1=5)              ; If variable 1 = 5, then execute commands
                        ; between IF and ELSE.
                        ; Otherwise, execute commands between ELSE and NIF
    GOTO pick1          ; Goto label pick1
  ELSE                  ; Else part of IF statement
    GOTO pick2          ; Goto label pick2
NIF                     ; End of IF statement
$ pick1                 ; Define label for pick1
  GO0011                ; Initiate motion on axes 3 and 4
  BREAK                 ; Break out of current subroutine or program
$ pick2                 ; Define label for pick2
  GO1001                ; Initiate motion on axes 1 and 4
END                     ; End definition of program pick
```

Conditional Looping and Branching

Conditional looping (`REPEAT`/`UNTIL` and `WHILE`/`NWHILE`) entails repeating a set of commands until or while a certain condition exists. In *conditional branching* (`IF`/`ELSE`/`NIF`), a specific set of commands is executed based on a certain condition. Both rely on the fulfillment of a conditional *expression*, a condition specified in the `UNTIL`, `WHILE`, or `IF` commands.

A `WAIT` command pauses command execution until a specific condition exists.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Flow Control Expression Examples

This section provides examples of expressions that can be used in conditional branching and looping commands (UNTIL, WHILE, and IF) and the WAIT command. These expressions can be constructed, in conjunction with relational and logical operators, with the following operands:

- Numeric variables and binary variables
- Inputs and outputs
- Current motion parameters and status
- Current motor & encoder position (steppers)
- Current commanded and actual position (servos)
- Error, axis, and system status
- Timer and counter values
- Data read from the serial port (stand-alone)
- Data read from the RP240 (stand-alone)
- Following conditions

Numeric and Binary Variables

A numeric variable (VAR) can be used within an expression if it is compared against another numeric variable, a value, or one of the comparison commands (see list on page 7). Note that not all of the comparison commands apply to every 6000 controller. When comparing a variable against another value, variable, or comparison command, the relational operators (=, >, >=, <, <=, <>) and logical operators (AND, OR, NOT) are used.

Expression	Description
(VAR1<VAR2)	True expression if variable 1 is less than variable 2
(VAR1>=2500)	True expression if variable 1 is greater than or equal to 2500
(VAR1=1AD)	True expression if variable 1 is equal to the decel of axis 1
(VAR1<VAR2 AND VAR4>1PE)	True expression if variable 1 is less than variable 2 and variable 4 is greater than the value of encoder 1

A binary variable (VARB) can be used within an expression, if the variable is compared against another binary variable, or a value. When comparing a variable against another value or variable, the relational operators (=, >, >=, <, <=, <>) and logical operators (AND, OR, NOT) are used.

Expression	Description
(VARB1<>VARB2)	True expression if binary variable 1 is not equal to binary variable 2
(VARB1=b1101 X111)	True expression if binary variable 1 is equal to 1101 X111
(VARB1<VARB2 AND VARB4>hF)	True expression if binary variable 1 is less than binary variable 2 and binary variable 4 is greater than the hexadecimal value of F

Inputs and Outputs

An input or output operand (IN, INO, LIM, OUT) can be used within an expression, if the operand is compared against a binary variable or a binary or hexadecimal value. When making the comparison, the relational operators (=, >, >=, <, <=, <>) and logical operators (AND, OR, NOT) are used.

Expression	Description
(IN.12=b1)	True expression if input 12 is equal to 1
(LIM>h3)	True expression if limit status is greater than hexadecimal 3

Current Motion Parameters and Status

Motion parameters consist of A, AD, D, V, VEL, status MOV. The motion parameters can be used within an expression, if the operand is compared against a numeric variable or value. The motion status operand must be compared against a binary variable or a binary or hexadecimal value. When making the comparison, the relational operators (=, >, >=, <, <=, <>) and logical operators (AND, OR, NOT) are used. (Following conditions are addressed below.)

Expression	Description
(VAR1<1VEL)	True expression if the value of variable 1 is less than the commanded velocity of axis 1
(1AD=25000)	True expression if axis 1 deceleration equals 25000
(MOV=b00)	True expression if moving status equals ØØ (axes 1 & 2 are not moving)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Current Motor and Encoder Position ***(Stepper Products Only)***

The current motor and encoder positions (PCE, PCM, PE, PER, PM, PMAS, PSHF, PSLV) can be used within an expression, if the operand is compared against a numeric variable or value. When making the comparison, the relational operators (=, >, >=, <, <=, <>) and logical operators (AND, OR, NOT) are used.

Expression	Description
(VAR1<1PM)	True expression if VAR1 is < commanded motor position of axis 1
(2PE=25000)	True expression if axis 2 encoder position equals 25000

Current Commanded & Actual Position ***(Servo Products Only)***

The current commanded and feedback device positions (ANI, CA, DAC, FB, LDT, PANI, PC, PCA, PCC, PCE, PCL, PER, PE, PMAS, PSHF, PSLV) can be used within an expression, if the operand is compared against a numeric variable or value. When making the comparison, the relational operators (=, >, >=, <, <=, <>) and logical operators (AND, OR, NOT) are used.

Expression	Description
(VAR1<1FB)	True expression if the value of variable 1 is less than the actual position (position of the assigned feedback device) of axis 1
(2PC=4000)	True expression if axis 2 commanded position equals 4000

Error, Axis, and System Status

The error status, axis status, and system status operands (ER, AS, SS) can be used within an expression, if the operand is compared against a binary variable or a binary or hexadecimal value. When making the comparison, the relational operators (=, >, >=, <, <=, <>) and logical operators (AND, OR, NOT) are used. Refer to page 232 for a list of status bit functions.

Expression	Description
(ER.12=b1)	True expression if error status bit 12 is equal to 1
(AS=h3FFD)	True expression if axis status is equal to hexadecimal 3FFD

Timer and Counter Values ***(Counter available on stepper products only)***

The current timer and counter values (TIM and CNT) can be used within an expression, if the operand is compared against a numeric variable or value. When making the comparison, the relational operators (=, >, >=, <, <=, <>) and logical operators (AND, OR, NOT) are used.

Expression	Description
(VAR1<TIM)	True expression if the value of variable 1 is less than the timer value
(1CNT>23567)	True expression if the value of counter #1 is greater than 23567

Data Read from the Communications Port

The READ command can be used to input data from the RS-232C serial port or the PC bus into a numeric variable. After the data has been read into a numeric variable, that variable may be used in an expression.

Example	Description
VARS8="ENTER DATA"	Define message (string variable 8)
VAR2=READ8	Send message (string variable 8) and then wait for immediate data to be read (into numeric variable 2)
!'88.3	Immediate data input (must type !' before the numeric value)
IF (VAR2<=100)	Evaluate expression to see if data read is < or equal to 100
. . . . .	
NIF	End of IF

Data Read from the RP240 ***(Stand-alone products only)***

The DREAD and DREADF commands can be used to input data from the RP240 into a numeric variable. DREAD reads a number from the RP240's numeric keypad. DREADF reads a number representing a RP240 function key. After the data has been read into a numeric variable, that variable may be used in an expression.

```
DCLEAR0             ; Clear RP240 display
DWRITE"HIT F4"      ; Send message to RP240 display
VAR3=DREADF         ; Read data from a RP240 function key into
                    ; numeric variable 3
IF (VAR3<>4)        ; Evaluate expression to see if function key F4 was hit
DCLEAR2             ; Clear RP240 display line 2
DWRITE"TRY AGAIN"   ; Send message to RP240 display
NIF                 ; End of IF
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

RP240 Data Read Immediate Mode ***(Stand-alone products only)***

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. Refer to the DREAD and DREADF command descriptions for programming examples.

NOTES

- While in the Data Read Immediate Mode, data is read into numeric variables only (VAR).
- This feature is not designed to be used in conjunction with the RP240's standard menus; 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.

Following Conditions

These Following conditions are available for conditional expressions: Axis status bit #26 (AS.26), Error status bit #14 (ER.14), Following status (FS), NMCY, PMAS, PSHF, PSLV, and VMAS.

Expression	Description
(2AS.26=b1)	True if a new motion profile on axis 2 is waiting for the GOWHEN condition to be true or a TRGFNc1xxxxxxx trigger.
(1ER.14=b1)	True if the GOWHEN condition on axis 1 is already true when the subsequent GO, GOL, FSHFC, or FSHFD command is executed.
(3FS.7=b0)	True if the master for slave axis 3 is in motion.
(2NMCY>200)	True if the master for axis 2 has moved through 200 cycles.
(1PMAS>12)	True if the master for axis 1 has traveled more than 12 units.
(1PSHF>1.5)	True if slave axis 2 has shifted more than 1.2 units.
(3PSLV>24)	True if slave axis 3's commanded position is more than 24 units.
(1VMAS<2)	True if the velocity of the master for axis 1 is less than 2 units/sec.

Conditional Looping

The 6000 controller supports two conditional looping structures—REPEAT/UNTIL and WHILE/NWHILE.

All commands between REPEAT and UNTIL are repeated until the expression contained within the parenthesis of the UNTIL command is true. The example below illustrates how a typical REPEAT/UNTIL conditional loop works. In this example, the REPEAT loop will execute 1 time, at which point the expression stated within the UNTIL command will be evaluated. If the expression is true, command processing will continue with the first command following the UNTIL command. If the expression is false, the REPEAT loop will be repeated.

```
VAR5=0            ; Initializes variable 5 to 0
DEL prog10        ; Delete a program before defining it
DEF prog10        ; Defines program prog10
INFNC1-A          ; Input 1 is not assigned a function, used with IN
INFNC2-A          ; Input 2 is not assigned a function, used with IN
INFNC3-A          ; Input 3 is not assigned a function, used with IN
INFNC4-A          ; Input 4 is not assigned a function, used with IN
OUTFNC1-A         ; Output 1 is programmable
A50               ; Acceleration is 50
AD50              ; Deceleration is 50
V5                ; Sets velocity to 5
D25000            ; Distance is 25,000
REPEAT            ; Begins the REPEAT loop
  GO1             ; Executes the move (Go)
  VAR5=VAR5+1     ; Variable 5 counts up from 0
UNTIL(IN=b1110 OR VAR5>10) ; When the inputs 1-4 are 1110, respectively or
                  ; VAR5 is greater than 10, the loop will stop.
OUT1              ; Turn on output 1 when finished with REPEAT loop
END               ; End program definition
RUN prog10        ; Initiate program prog10
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

All commands between WHILE and NWHILE are repeated as long as the WHILE condition is true. The following example illustrates how a typical WHILE/NWHILE conditional loop works. In this example, the WHILE loop will execute if the expression is true. If the expression is false, the WHILE loop will not execute.

```
VAR5=0           ; Initializes variable 5 to 0
DEL prog10       ; Delete a program before defining it
DEF prog10       ; Defines program prog10
INFNC1-A         ; Input 1 is not assigned a function, used with IN
INFNC2-A         ; Input 2 is not assigned a function, used with IN
INFNC3-A         ; Input 3 is not assigned a function, used with IN
INFNC4-A         ; Input 4 is not assigned a function, used with IN
OUTFNC1-A        ; Output 1 is programmable
A50              ; Acceleration is 50
AD50             ; Deceleration is 50
V5               ; Sets velocity to 5
D25000           ; Distance is 25,000
WHILE(IN=b1110 OR VAR5>10) ; While the inputs 1-4 are 1110, respectively or
                 ; VAR5 is greater than 10, the loop will continue.
  GO1            ; Executes the move (Go)
VAR5=VAR5+1      ; Variable 5 counts up from 0
NWHILE           ; End WHILE command
OUT1             ; Turn on output 1 when finished with WHILE loop
END              ; End program definition

; ************************************************
; * To run prog10, execute the "RUN prog10" command     *
; ************************************************
```

Conditional Branching

You can use the IF command for conditional branching. All commands between IF and ELSE are executed if the expression contained within the parentheses of the IF command is true. If the expression is false, the commands between ELSE and NIF are executed. If the ELSE is not needed, it may be omitted. The commands between IF and NIF are executed if the expression is true. Examples of these commands are as follows.

```
DEL prog10       ; Delete a program before defining it
DEF prog10       ; Defines program prog10
INFNC1-A         ; Input 1 is not assigned a function, used with IN
INFNC2-A         ; Input 2 is not assigned a function, used with IN
INFNC3-A         ; Input 3 is not assigned a function, used with IN
INFNC4-A         ; Input 4 is not assigned a function, used with IN
A50              ; Acceleration is 50
AD50             ; Deceleration is 50
V5               ; Sets velocity to 5
IF(VAR1>0)       ; IF variable 1 is greater than zero
   D25000        ; Distance is 25,000
   ELSE          ; Else
   D50000        ; Distance is 50,000
NIF              ; End if command
IF(IN=b1110)     ; If inputs 1-4 are 1110, initiate axis 1 move
   GO1           ; Executes the move (Go)
NIF              ; End IF command
END              ; End program definition

; ************************************************
; * To run prog10, execute the "RUN prog10" command     *
; ************************************************
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Program Interrupts *(ON Conditions)*

While executing a program, the 6000 controller can interrupt the program based on several possible *ON conditions*: programmable input(s) status, user status, or the value of numeric variables #1 or #2. These ON conditions are enabled with the ONCOND command, and are defined with the commands listed below. After the ON conditions are enabled (with the ONCOND command), an ON condition interrupt can occur at any point in program execution. When an ON condition occurs, the controller performs a GOSUB to the program assigned as the ON program and then passes control back to the original program and resumes command execution at the command line from which the interruption occurred.

Within the ON program, the programmer is responsible for checking which ON condition caused the branch (if multiple ON conditions are enabled with the ONCOND command). Once a branch to the ON program occurs, the ON program will not be called again until after it has finished executing. After returning from the ON program, the condition that caused the branch must evaluate false before another branch to the ON program will be allowed.

SETUP FOR PROGRAM INTERRUPT (see programming example below)

1. Defined a program to be used as the ON program to which the controller will GOSUB when an ON condition evaluates true.
2. Use the ONP command to assign the program as the ON program.
3. Use the ONCOND command to enable the ON conditions that you are using. The syntax for the ONCOND command is ONCOND<b><b><b><b>, where the first <b> is for the ONIN condition, the second for ONUS, the third for ONVARA, and the fourth for ONVARB.

ON conditions:

ONIN ..... Specify an input bit pattern that will cause a GOSUB to the program assigned as the ON program (see programming example below).

ONUS ..... Specify an user status bit pattern that will cause a GOSUB to the ON program. The user status bits are defined with the INDUST command.

ONVARA ..... Specify the range of numeric variable #1 (VAR1) that will cause a GOSUB to the ON program. For example, ONVARAØ,2Ø establishes the condition that if the value of VAR1 is ≤0 or ≥20, the ON program will be called.

ONVARB ..... This is the same function as ONVARA, but for numeric variable #2 (VAR2)

Programming Example: Configures the controller to increment variable #1 when input #1 goes active. If input #1 does go active, control will be passed (GOSUB) to the ON program (onjump), the commands within the ON program will be executed, and control will then be passed back to the original program.

```
DEF onjump      ; Begin definition of program onjump
VAR1=VAR1+1     ; Increment variable 1
END             ; End definition of program onjump

VAR1=0          ; Initialize variable 1
ONIN1           ; When input 1 becomes active, branch to the ON program
ONP onjump      ; Assign the onjump program as the ON program
ONCOND1000      ; Enable only the ONIN function.  Disable the ONUS, ONVARA,
                ; and ONVARB functions, respectively
```

Situations in which 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 motion is in progress due to GO, GOL, GOWHEN, HOM, JOY, JOG, or PRUN and the continuous command execution mode is disabled (COMEXCØ).
- 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

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Error Handling

The 6000 Series products have the ability to detect and recover the following error conditions:

- Steppers Only: Stall detected on any axis (error bit #1) -- not applicable to OEM-AT6400
- Hardware end-of-travel limit encountered on any axis (error bit #2)
- Software end-of-travel limit encountered on any axis (error bit #3)
- Drive fault input activated any axis (error bit #4)
- Commanded kill or stop (error bit #5)
- Kill input activated (error bit #6)
- User fault input activated (error bit #7)
- Stop input activated (error bit #8)
- Steppers Only: Pulse cut-off (**PCUT**) input not grounded (error bit #9)
  Servos Only: Enable (**ENBL**) input not grounded (error bit #9)
- Profile for a pre-emptive GO or a registration move is not possible (error bit #10)
- Servos Only: Target zone settling timeout (error bit #11)
- Servos Only: Allowable position error (SMPER) exceeded (error bit #12)
- Servos Only: GOWHEN condition already true when the subsequent GO, GOL, FSHFC, or FSHFD command was executed (error bit #14)
- Hydraulic Servos Only: LDT position read error (error bit #15)

DEBUG TOOLS
For information on program debug tools, refer to page 231.

Enabling Error Checking

To detect and respond to the error conditions noted above, the corresponding error-checking bit(s) must be enabled with the ERROR command (refer to the *ERROR Bit #* column in the table below). If 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.

For example, if you wish the 6000 controller to branch to the error program when a hardware end-of-travel limit is encountered (error bit #2) or when a drive fault occurs (error bit #4), you would issue the ERROR0101 command to enable error-checking bits #2 and #4.

☞ Helpful Hint: Within your program structure, you can use the IF and ER commands to conditionally enable the error-checking bits that will in turn call the ERRORP program (refer to the programming example below).

Defining the Error Program

The purpose of the error program is to provide a programmed response to certain error conditions (see list above) 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. Refer to the error program set-up example below.

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. To un-assign a program from being the error program, issue the ERRORP CLR command (e.g., as in this example, it does not delete the CRASH program, but merely unlinks it from its assignment as the error program).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Canceling the Branch to the Error Program

If 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. 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, **PCUT** or **ENBL** input activated, in which case the error program is called only once.)

There are three options for canceling the branch to the error program:

- Disable the error-checking bit with the ERROR.n-Ø 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-Ø 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 Error Program	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-AT6400*	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 (enable Input Functions with INFEN1; and set Drive Fault Level with DRFLVL) *n/a to OEM-AT6400*	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.
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
9	Steppers: **P-CUT** input not grounded Servos: **ENBL** input not grounded	Goto	Re-ground the **P-CUT** input (steppers) or **ENBL** input (servos), and issue a DRIVE1111 command.
10	Profile for pre-emptive GO or registration move not possible	Gosub	Issue another GO command.
11	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 was already true when the subsequent GO, GOL, GSHFC, or FSHFD command was executed.	Goto	Issue a !K command and check the program logic. Use the TRACE and STEP features if necessary.
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**.
16	Bad command was detected	Gosub	Issue the TCMDER command to I.D. the command.

Reserved Bits: Bits 13, and 17 - 32.

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. To prevent a return to the point at which the error occurred, use the HALT command to end program execution or 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.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Error Program Set-up Example

The following is an example of how to set up an error program. This particular example is for handling the occurrence of a user fault.

Step 1 Create a program file (in Motion Architect's Editor module) to set up the error program:

```
; ************************************************************************
; *  Assign the user fault input function to programmable input #1.      *
; *  The purpose of the user fault input is to detect the occurrence of  *
; *  a fault external to the 6000 controller and the motor/drive.        *
; *  This input will generate an error condition.                        *
; ************************************************************************
INFNC1-F        ; Define programmable input #1 as a user fault input
INFEN1          ; Enable input functions (For the purposes of this
                ; set-up example, make sure programmable input #1 is
                ; not activated.)

; ************************************************************************
; *  Define a program to respond to the user fault situation (call the   *
; *  program fault), and then assign that program as the error program.  *
; *  The purpose of the fault program is to display a message to         *
; *  inform the operator that the user fault input has been activated.   *
; ************************************************************************
DEL fault       ; Delete a program before defining it (a precaution)
DEF fault       ; Begin definition of program fault
IF(ER.7=b1)     ; Check if error bit 7 equals 1
                ; (which means the user fault input has been activated)
WRITE"FAULT INPUT\10\13"    ; Send the message FAULT INPUT
T3              ; Wait 3 seconds
NIF             ; End IF command
END             ; End definition of program fault
ERRORP fault    ; Assign the program called fault as the error program

; ************************************************************************
; *  Enable the user fault error-checking bit by putting a "1" in the    *
; *  seventh bit of the ERROR command.  After enabling this              *
; *  error-checking bit, the controller will branch to the error         *
; *  program whenever the user fault input is activated.                 *
; ************************************************************************
ERROR0000001    ; Branch to error program upon user fault input (As an
                ; alternative to the ERROR0000001 command, you could also
                ; enable bit #7 by issuing the ERROR.7-1 command.)
```

Step 2 Save the program file in the Editor module. Then, using the Terminal module, download the program file to the 6000 controller.

Step 3 Test the error handling:

1. While in the terminal emulator, enter these four commands:

```
L                       ; Loop command
WRITE"IN LOOP\10\13"    ; Send Message "IN LOOP" to the terminal display
T2                      ; Wait 2 seconds
LN                      ; End the loop ("IN LOOP" will be displayed
                        ; once every 2 seconds)
```

2. While the `IN LOOP` loop is executing in the terminal emulator, enter the `!INEN1` command. The `!INEN1` command disables input #1 and forces it on for testing purposes. This simulates the physical activation of input #1. (Since the error program is called continuously until the branch to the error program is canceled, the message `FAULT INPUT` will be repeatedly displayed once every 3 seconds.)

3. While the `FAULT INPUT` loop is executing in the terminal emulator, enter the `!INENE` command. The `!INENE` command re-enables input #1. The message `IN LOOP` will not be displayed again, because the user fault input error is a GOTO branch (not a GOSUB branch) to the error program.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Non-Volatile Memory (Stand-Alone Products Only)

When using stand-alone serial-based 6000 controllers the items listed below are automatically stored in non-volatile memory (battery-backed RAM). Cycling power or issuing a RESET command will not affect these settings.

Item	610n	615n	620n	625n	6270
Absolute position reference (PSET)	n/a	n/a	n/a	n/a	•
Commanded direction polarity (CMDDIR)	•	•	•	n/a	•
Compiled profiles (PCOMP) – see note below *	•	•	•	•	•
Device address (ADDR)	•	•	•	•	•
Feedback device polarity					
ANI polarity (ANIPOL)	n/a	n/a	n/a	n/a	•
Encoder polarity (ENCPOL)	•	•	•	n/a	•
LDT polarity (LDTPOL)	n/a	n/a	n/a	n/a	•
LDT gradient	n/a	n/a	n/a	n/a	•
Memory allocation (MEMORY)	•	•	•	•	•
Motor inductance (DMTIND)	•	n/a	n/a	n/a	n/a
Motor static torque (DMTSTT)	•	n/a	n/a	n/a	n/a
Power-up program (STARTP)	•	•	•	•	•
Programs (defined with DEF & END)	•	•	•	•	•
RP240 check (DRPCHK)	•	•	•	•	n/a
RP240 password (DPASS)	•	•	•	•	•
RS-232C baud rate	•	•	•	•	•
Servo gain sets (SGSET)	n/a	•	n/a	•	•
Variables: (VAR, VARB, and VARS)	•	•	•	•	•

* Compiled contours are always saved in the Compiled portion of batter-backed RAM. However, compiled individual axis profiles (GOBUF profiles) are removed from Compiled memory if you run them with the PRUN command and later cycle power or RESET the controller (you will have to re-compile them with the PCOMP command).

A checksum is calculated for the non-volatile memory area each time you power up or reset your 6000 controller. A bad checksum indicates that the user memory has been corrupted (possibly due to electrical noise) or has been cleared (due to a spent battery). The controller will clear all user memory when a bad checksum is calculated on power up or reset, and bit 22 will be set in the TSS command response.

System Performance

Several commands (listed below), when enabled, will slow command processing. This degradation in performance will not be noticeable for most applications. But for some, it may be necessary to disable one or all of these commands.

- SCALE (enable/disable scaling)
- INDUSE (enable/disable user status updates)
- INFEN (enable/disable drive fault and programmable input functions)
- OUTFEN (enable/disable programmable output functions)
- ONCOND (enable/disable ON conditions)

Servo Products: Changing the INDAX and/or SSFR command values affects the servo sampling update, the motion trajectory update, and the *system update rate*. The system update rate is the rate for I/O updates, input debounce, timer resolution, fast status update (bus-based products), and LDT position update (6270). For more information, refer to the SSFR command description.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

CHAPTER TWO

2 Communication

IN THIS CHAPTER

This chapter will help you understand these aspects of communicating with your 6000 Series product:

- Motion Architect™ communication features.....36
- DOS support software for stand-alone products 37
- DOS support software for bus-based products 38
- PC-AT bus communication registers — *bus-based controllers only* 43
- DDE6000™ (Dynamic Data Exchange server for 6000 products) 50
- Dynamic Link Library (DLL) — *bus-based controllers only*.....51
- Motion OCX Toolkit™ (OCX controls) — *bus-based controllers only*.....62
- PC-AT interrupts — *bus-based controllers only*.....63
- Controlling multiple serial ports — *stand-alone controllers only*.....70
- RS-232C daisy-chaining — *stand-alone controllers only*.....72
- RS-485 multi-drop — *stand-alone controllers only*.....75

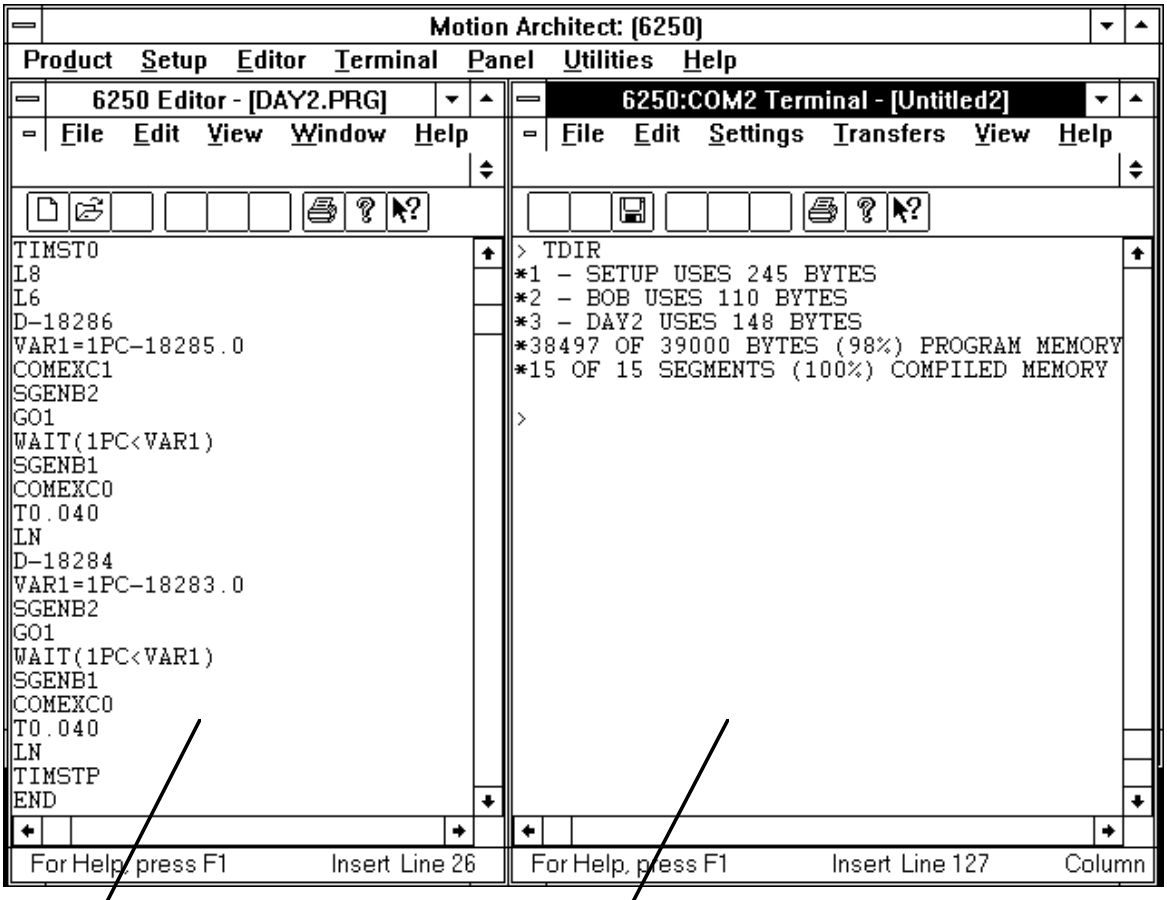
Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Motion Architect provides easy terminal emulation support in the Terminal, Panel, and Controller Tuner modules:

- Terminal emulator window.
- Communication setup parameters (board address and interrupts for bus-based controllers, and COM port selection for stand-alone controllers).
- Download the bus-based product's soft operating system (prerequisite to connecting, programming, or sending/receiving motion programs).
- Download motion programs (file from your hard drive) to the controller. The program is immediately executed with it is received by the controller.
- Upload motion programs from the controller.

For more in-depth user information, refer to Motion Architect's online help system.

TIP: Try re-sizing the Editor and Terminal windows to be viewed side by side (see example below). In doing so you can quickly jump back and forth between editing a program and downloading it to the product and checking programming responses and error messages. *The program development scenario on page 8 uses this side-by-side technique.*



Program Editor: Create and edit programs, save them, and then download them from the Terminal module.

Terminal Emulator: Communicate directly with the 6000 controller. Download files containing stand-alone commands and/or complete programs or subroutines. Check system responses. Upload programs from the 6000 controller.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

DOS Support Software for Stand-Alone Products

The *6000 DOS Support Disk*, which provides a program for RS-232C terminal emulation and program editing for stand-alone products, is available from your local ATC or distributor (or contact Compumotor at the numbers listed on the inside cover of this manual).

This program is designed to communicate to a Compumotor 6000 Series stand-alone product via your computer's RS-232C port.

Installing and Running the Program

Install……Place the disk in drive A. Change to drive A by typing "A:". At the DOS prompt for drive A type "INSTALL".

Run……Type "6000" at the DOS prompt.

Pressing the ENTER key will move you down a level into the program, and pressing the ESCAPE key will move you up a level. At any point in the program you can get help information by pressing F1.

The main menu gives you a choice of selecting **Editor**, **Terminal Emulator**, **Set-Up**, or **Quit**. The choices are described below.

Menu Item	Description
Editor	Allows you to create motion control programs for use in your 6000 Series product. Programs can be stored to disk and recalled from disk. Programs may be downloaded to the 6000 Series product. **F1 - Help**: Help is always available by pressing F1. **F2 - Upload**: Upload a program from a 6000 Series product. **F3 - Download**: Download a program to a 6000 Series product. **F4 - File**: • Recall a stored program from disk • Save the editor's contents to disk • Delete a program • Insert a stored program into the editor at the cursor position • Print the editor's contents • Specify the current working directory for the above file operations • Specify a file pattern for the above file operations **F5 - Commands**: 6000 Series programming commands and their descriptions. **F6 - Clear**: Clear the editor's contents.
Terminal Emulator	Puts the computer in communication with a 6000 Series product. The computer is emulating a *dumb terminal*. When in this mode, commands entered are directly sent to the 6000 Series product. The *6000 DOS Support Program* requires a 3-wire RS-232C interface (Rx, Tx, and GND). When entering the Terminal Emulator, the RS-232C interface is automatically verified. If the interface is not functional, go to **Set-Up** to set the proper RS-232C parameters and to verify the interface.
Set-Up	Allows modification of important Editor and Terminal Emulator parameters: • Editor Size (64 K) • Baud Rate (default for 6000 Series products is 9600) • Data Bits (set to 8) • Parity (set to NONE) • Stop Bits (set to 1) • COM1/COM2 (select a communication port) • Command Delay (adjustable delay between commands from the Editor's Download function) • Check Out (test for proper serial connection to the 6000 series product)
Quit	**Quit** the program and return to DOS.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

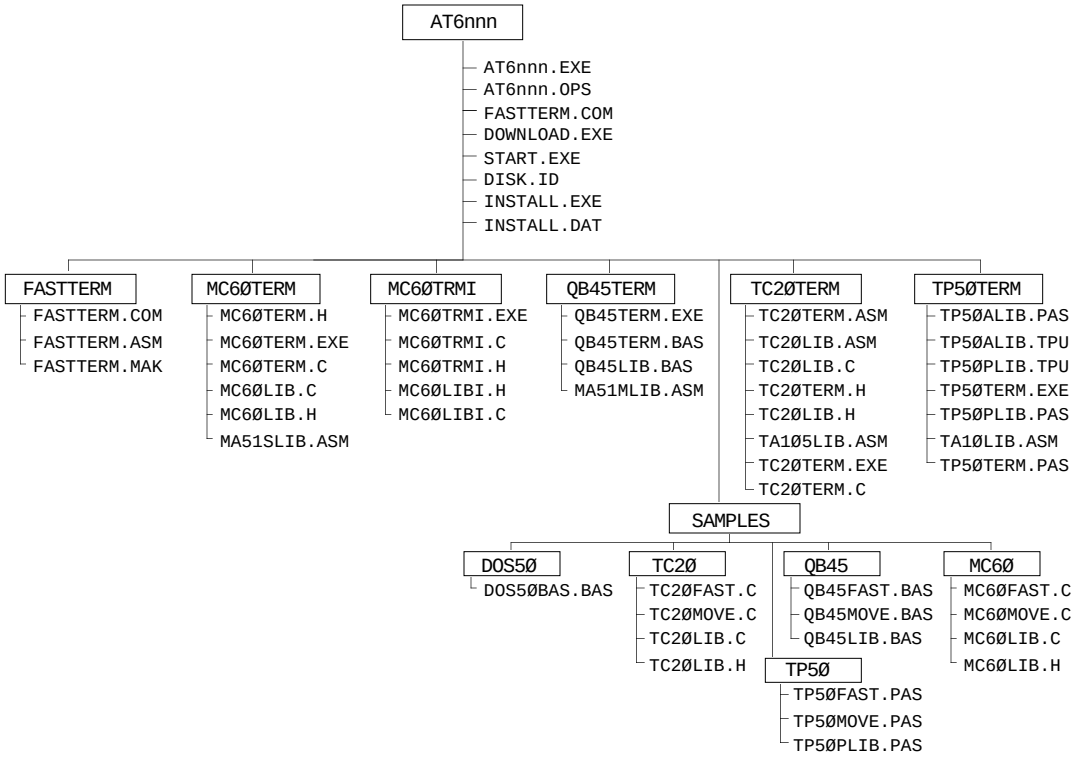
NOTE

This section uses a generic reference ("AT6nnn") to represent all 6000 Series bus-based products. When referring to the file names and programming examples, substitute the name of your product where you read "AT6nnn". For example, if you are using the AT6450, type "AT6450".

Exceptions:

- **OEM products**: Use the root name of the product (e.g., if using an OEM-AT6400, type "AT6400").
- **AT6200 and OEM-AT6200**: Type "AT6400".

Bus-based products are shipped with a DOS support disk (see diskette labeled with the product's name). Upon installation, the support disk is divided into seven sub-directories, in addition to the root directory (see illustration below).



The root directory contains all of the operating system files required by the bus-based controller, and an installation routine which will copy all the files from the support diskette to your computer's hard drive.

Stepper products only: The root directory on the DOS support disk also contains a test program (TEST.EXE) designed to help you test most of the motion and I/O capabilities of the stepper controller. Refer to your controller's installation guide for instructions.

The sub-directories contain programs that demonstrate communication with the controller using different programming languages. The languages used in each sub-directory are listed below.

Sub-directory	Language
FASTTERM	Microsoft ASSEMBLY 5.1
MC60TERM	Microsoft C 6.0
MC60TRMI	Microsoft C 6.0 (Demonstrates Interrupts)
QB45TERM	Microsoft QuickBASIC 4.5
SAMPLES	QuickBASIC 4.5, Microsoft C 6.0, Turbo C 2.0, PASCAL 5.0
TC20TERM	Borland Turbo C 2.0
TP50TERM	Borland Turbo PASCAL 5.0

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Downloading the Operating System

Before you can use the bus-based controller within your application you must first download the operating system (6000 Series Command Language). The file called AT6nnn.OPS on the supplied DOS Support Software Diskette contains the operating system. Use the file called AT6nnn.EXE to download the operating system.

As an example, to download the controller's operating system, type AT6nnn at the DOS prompt and the operating system (AT6nnn.OPS) will be downloaded from the PC-AT to I/O address 768 (300H), the default address for bus-based controllers.

If the controller is addressed other than the default (768), then a command switch is required. Instead of typing AT6nnn, type AT6nnn /port=address, where the word "address" is replaced by the address value that you set with the DIP switch on the controller's PC-AT card (refer to your product's ***Installation Guide*** for optional DIP switch settings). For example, if you set the address to decimal 800 (320H), then you would download the operating system by typing AT6nnn /port=800.

If you would like to suppress the messages displayed when downloading the operating system, type AT6nnn /quiet.

If you would like to pause after downloading the operating system and wait for an operator to press any key, type AT6nnn /pause.

If the operating system is not located in the default directory or in the same directory as the AT6nnn.EXE file, the path is not searched; this would require you to specify the path of the operating system on the command line. The path must be specified before any other command switches. Instead of typing AT6nnn, type AT6nnn \pathname\AT6nnn.OPS. For example, if the operating system was located in the sub-directory project1, then type AT6nnn \project1\AT6nnn.OPS.

Downloading Error

If the operating system does not download properly, an error message will be displayed. This error can result from an invalid address or an incompatible transfer mode. *Downloading error codes are listed on page 243.*

If the error is a result of an invalid address, verify the address DIP switches on the controller's AT card (refer to your product's ***Installation Guide*** for details).

If the error is a result of an incompatible transfer mode, change DIP switch #8 on DIP switch bank #1 on the controller's AT card from ON (16-bit mode) to OFF (8-bit mode). The 8-bit mode may be required for reliable bus communications. By switching to the 8-bit mode, communication between the controller and the PC-AT will be slowed. Refer to your product's ***Installation Guide*** for details.

Downloading Methods

There are three ways to download the operating system within your application. The operating system can be downloaded upon system power-up, from a batch file, or from the application program itself.

Download upon System Power-up

From within the AUTOEXEC.BAT file, execute the AT6nnn.EXE file. This will require specifying the path in which the AT6nnn.EXE file is located. Refer to your DOS manual for more information on the AUTOEXEC.BAT file.

For example, the INSTALL.EXE program provided with the DOS Support Disk places the AT6nnn.EXE file in the sub-directory AT6nnn. The following statement would be placed in the AUTOEXEC.BAT file: \AT6nnn\AT6nnn.EXE \AT6nnn\AT6nnn.OPS /port=768.

Download from a Batch File

Create a batch file that contains the command to download the AT6nnn operating system (AT6nnn) and the application program name itself. Refer to your DOS manual for more information on batch file operations.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

This section describes how to download from C and PASCAL programs.

Downloading from a C Program: To download the operating system from a C program, a *child* process must be created. The C command `spawnl` is used to create a child process in which the operating system is downloaded (see code example below).

```
#include <stdio.h>
#include <process.h>

void main()
{
  int error_code = 0;
  char *pathname = "C:\\AT6nnn\\AT6nnn.EXE";
  char *args[] = {    "AT6nnn.EXE",
                      "C:\\AT6nnn\\AT6nnn.OPS",
                      "/port=768",
                      "/quiet",
                      NULL
                 };

  error_code=spawnl(P_WAIT,pathname,args[0],args[1],args[2],args[3],NULL);
  if(error_code==-1) {
       printf("Could not locate AT6nnn operating system\n");
       exit(0);
  }
  else if (error_code > 0) {
       printf("Failed to download AT6nnn operating system\n");
       exit(0);
  }
}
```

Downloading from PASCAL: To download the operating system from a PASCAL program, a separate process must be executed. The PASCAL command `exec` is used to run a separate process in which the operating system is downloaded (see code example below).

```
Program download;

uses Crt;

begin
  swapvectors;
  exec('\AT6nnn\AT6nnn.EXE','AT6nnn  \AT6nnn\AT6nnn.OPS /port=768 /quiet');
  swapvectors;
  if Doserror <> 0 then
    begin
      writeln('Dos error #',Doserror);
    end;
end.
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Terminal Emulation

Once the operating system has been downloaded, an application program can be run on the controller. The DOS Support Disk provides six terminal emulator programs to allow direct communication with the controller (see table below).

Directory	Terminal Emulator Program Name
FASTTERM	FASTTERM.COM
MC60TERM	MC60TERM.EXE
MC60TRMI	MC60TRMI.EXE
QB45TERM	QB45TERM.EXE
TC20TERM	TC20TERM.EXE
TP50TERM	TP50TERM.EXE

To initiate any one of the terminal emulator programs, simply change to the directory that contains the terminal emulator program you wish to use, and type the name of the terminal emulator program at the DOS prompt.

All of the terminal emulator programs, except MC60TRMI.EXE, provide the same interface to the controller, and are all written in different programming languages. MC60TRMI.EXE provides a terminal emulator interface to the controller; however, this program takes advantage of the controller's interrupt capability. To use a terminal emulator program (i.e. FASTTERM.COM, MC60TERM.EXE, etc.) to download a text file containing the 6000 Series commands, simply invoke the terminal emulator program and press function key 1 (F1).

Downloading Application Programs from the DOS prompt (C:\>)

To use the DOWNLOAD.EXE program, place the display option, the address, and the text file name on the command line (syntax is DOWNLOAD1 768 *textfile.nme*). For a description of the display options, type DOWNLOAD without any command line arguments.

Once a text file has been downloaded into the controller, the programs contained within the text file can be initiated by either typing the name of the program when in a terminal emulator, or by using the START.EXE program contained on the DOS Support Disk.

Example

For example, assume the text file MYPROG.TXT contained these three 6000 language programs:

Program: apple

```
DEF apple
A10,10,10,100
V1,1,1,1
D20000,20000,60000,60000
GO1111
END
```

Program: pear

```
DEF pear
A100,100,100,100
V10,10,10,10
D1000,1000,500,500
GO1111
GO1100
GO0011
END
```

Program: peach

```
DEF peach
apple
pear
apple
pear
END
```

To initiate the program called peach after it has been downloaded via a terminal emulator, simply type peach within the terminal emulator.

To download and initiate the program peach from the command line, or from a batch file, simply state:

DOWNLOAD *<display option> <device address> textfile.1 textfile.2 textfile.3 ...*
START *<device address> program_name*

e.g. DOWNLOAD1 768 MYPROG.TXT
START 768 peach

Display Options (*<display option>* **)**

0 = No error messages will be displayed on screen.
1 = All commands and error messages will be displayed.
2 = Error messages only, if any, will be displayed.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Creating Your Own DOS-Based Application Program

Creating a program to control an application can often be difficult. To ease the programming burden, Compumotor has provided communication interface routines for ASSEMBLY, BASIC, C, and PASCAL. The routines are supplied in the seven sub-directories on the DOS Support Disk (see table below).

Directory	File to use	Function of the file
FASTTERM	FASTTERM.ASM	Shows how to communicate with the controller using ASSEMBLY.
MC60TERM	MC60TERM.C MC60LIB.C	Shows how to communicate with the controller using Microsoft C. The file MC60TERM.C shows how to use the subroutines created within MC60LIB.C.
TC20TERM	TC20TERM.C TC20LIB.C	Shows how to communicate with the controller using Borland Turbo C. The file TC20TERM.C shows how to use the subroutines created within TC20LIB.C.
TP50TERM	TP50TERM.PAS TP50PLIB.PAS	Shows how to communicate with the controller using Borland Turbo Pascal. The file TP50TERM.PAS shows how to use the subroutines created within TP50PLIB.PAS.
QB45TERM	QB45TERM.BAS QB45LIB.BAS	Shows how to communicate with the controller using Microsoft QuickBASIC. The file QB45TERM.BAS shows how to use the subroutines created within QB45LIB.BAS.
MC60TRMI	MC60TRMI.C MC60LIBI.C	Shows how to communicate with the controller using Microsoft C. This file also shows how to use the interrupt capability of the controller. The file MC60TRMI.C shows how to use the subroutines created within MC60LIBI.C.
SAMPLES		
DOS50	DOS50BAS.BAS	Shows how to communicate with the controller using DOS QuickBASIC.
TC20	TC20FAST.C TC20MOVE.C TC20LIB.C TC20LIB.H	Shows how to use the controller's fast status area (Borland Turbo C 2.0). Shows samples of basic moves (Borland Turbo C 2.0). Command routines for the controller (used with TC20MOVE.C) Command routines for the controller (used with TC20MOVE.C)
TP50	TP50FAST.PAS TP50MOVE.PAS TP50PLIB.PAS	Shows how to use the controller's fast status area (Borland Turbo Pascal 5.0). Shows samples of basic moves (Borland Turbo Pascal 5.0). Command routines for the controller (used with TP50MOVE.PAS)
QB45	QB45FAST.BAS QB45MOVE.BAS QB45LIB.BAS	Shows how to use the controller's fast status area (Microsoft QuickBASIC 4.5). Shows samples of basic moves (Microsoft QuickBASIC 4.5). Command routines for the controller (used with QB45MOVE.BAS)
MC60	MC60FAST.C MC60MOVE.C MC60LIB.C MC60LIB.H	Shows how to use the fast status area (Microsoft C 6.0). Shows samples of basic moves (Microsoft C 6.0). Command routines for the controller (used with MC60MOVE.C) Command routines for the controller (used with MC60MOVE.C)

The above sample programs ("SAMPLES" directory) utilize the functions SendAT6nnnBlock(param1,param2) and RecvAT6nnnBlock() to communicate with the AT6nnn product. The two parameters that are passed are the board address (in decimal) and a string. In SendAT6nnnBlock(address,command), the "command" is either a 6000 Language command string or a string variable representing a 6000 Language command line. In RecvAT6nnnBlock(address,response), the "response" is a string variable containing the response from the card.

Preventing the output buffer from filling up

It is important that the responses are read from the controller when sending commands; otherwise, the output buffer of the controller will fill up and the card will not accept any more commands until the output buffer is read via RecvAT6nnnBlock(). This can happen when the default error level, ERRLVL4, is set (all prompts and error messages are returned). Each time a valid command is received by the controller, a prompt (ERR0RK characters) is returned, indicating that the controller is ready for another command. If the output buffer is never read, it will eventually fill up with the ERR0RK characters. If you do not want to read a response each time a command is sent to the controller, then set the error level to ERRLVL1 or ERRLVL0. With these two settings, no error messages or prompts are generated by the controller, but bear in mind that all requested data (such as TPE, TAS, etc.) will be returned with minimal formatting.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

PC-AT Bus Communication Registers

Address (defaults in parenthesis)	Function
Base, Base+1	Read/write data to/from controller
Base (300h)	Write = Send data to controller (HB) Read = Receive data from controller (HB)
Base+1 (301h)	Write = Send data to controller (LB) Read = Receive data from controller (LB)
Base+2, Base+3 (302h,303h)	Read/Write Fast Status area
Base+2	Write = Set fast status pointer Read = Fast status byte
Base+3 (303h)	Write = *reserved* Read = Fast status byte
Base+4	Controller Status
Base+4 (304h)	Write = Set/clear status to controller Read = Read status from controller
Base+5, Base+6, Base+7	*Reserved*

Fast Status Register (Base+2, Base+3)

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 (see table on previous page).*

A fast status register is available to read various controller status data. The fast status 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).

The fast status register differs between stepper products and servo products. Each has eight 2-bytes blocks, but with differing information (see tables below). Another difference is that you may customize blocks 7 & 8 on stepper products, and blocks 3-8 on servo products (refer to *Customizing the Fast Status Register*, page 48, for details).

Fast Status, Steppers

	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 side note for *Limits Bit Assignments* *		1 word
	25	Other Input Status – see TINO		1 word
	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 (2ms timer, starts on computer powerup)*		1 word
	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

* Limits Bit Assignments:
Ø = axis 1 positive hard limit
1 = axis 1 negative hard limit
2 = axis 2 positive hard limit
3 = axis 2 negative hard limit
4 = axis 3 positive hard limit
5 = axis 3 negative hard limit
6 = axis 4 positive hard limit
7 = axis 4 negative hard limit
8 = axis 1 home limit
9 = axis 2 home limit
10 = axis 3 home limit
11 = axis 4 home limit

These bits report the current state of the input — not necessarily whether a hardware limit has been encountered.

* Timer starts on computer powerup; rolls over; updated once 2 msec update.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Fast Status, Servos

	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 Position (counts), axis 2	2 words
	0C	Feedback Device Position (counts), axis 3	2 words
	0E	Feedback Device Position (counts), axis 4	2 words
Block 3	10	Commanded Velocity (counts/sec), axis 1 – see TVEL	2 words
	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 side note *Limits Bit Assignments* on page 43	1 word
	25	Other Input Status – see TINO	1 word
	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 *	1 word
	2E	Programmable Timer Value – see TIMST	2 words
Block 7	30	ANI input value, input 1 (ADC counts; 819 counts/volt) – see TPANI	1 word
	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; 2048 counts/10 volts) – TDAC	1 word
	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

* Timer starts on computer powerup; rolls over; updated once per *system update* (see SSFR).

Reading The Fast Status Register

Below is a step-by-step procedure for reading information from the fast status area.

Step 1 Request that the information contained within the fast status register be updated. This is accomplished by writing the byte 48 Hex to the status register, located four address locations above the base address.

Step 2 Wait for the information in the fast status area to be updated. This is accomplished by reading the status register (bits 0 - 7) until bit 3 is set.

Step 3 Point to the information you wish to retrieve. This is accomplished by writing the hex offset value (1 byte) shown in the table above to the fast status register (2 address locations above the base address).

Step 4 Read the information from the fast status area, one word (two bytes) at a time.

As you read each word, (by writing to Base+2), the internal pointer will automatically increment to the next word in the block until a block boundary is reached. A block boundary occurs every 8 words, starting from the first word. Once a block boundary is reached, step 3 above must be repeated; otherwise, the pointer will wrap around to the beginning of the same segment.

If two words are required to be read in order to obtain all the information, the first word read is the most significant in terms of value.

Step 5 After the word(s) are read, each word must swap its most significant and least significant bytes. These steps are illustrated by the source code examples below.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Retrieving information from the fast status area using C: The source code example below is written in Turbo C. If you are using Microsoft C, replace "inport" with "inpw", replace "outport" with "outw", replace "inportb" with "inp", and replace "outportb" with "out". This example is also provided on the DOS Support Disk in sub-directory SAMPLES, files TC20FAST.C and MC60FAST.C.

```
  #include <stdio.h>
#define STATUS              4
#define FASTSTATUS          2
#define REQ_STATUS          0x48
#define CS_UPDATED          0x08
#define AXIS1_CMD           0x00
#define INO_STATUS          0x25
unsigned int address;         /* global address */
  void request_status(void)
  {
    outportb(address+STATUS, REQ_STATUS);  /* request fast status update */
    /* wait for fast status information to be updated */
    while(!(inportb(address+STATUS)&CS_UPDATED));
  }
  void set_pointer(int status_offset)
  {
    outportb(address+FASTSTATUS, status_offset);
  }
  void read_status( unsigned int * status_high; unsigned int * status_low;
                    unsigned long * status)
  {
    /* read fast status information */
    * status_high = inport(address+FASTSTATUS);
    * status_low = inport(address+FASTSTATUS);
    /* build status because of low/high ordering */
    * status_high = (* status_high<<8) + (* status_high>>8);
    * status_low = (* status_low<<8) + (* status_low>>8);
    * status = * status_high;
    * status = (* status<<16) + * status_low;
  }
  void main(void)
  {
    unsigned int word_high, word_low, ino_bits;
    unsigned long fast_status;
    char pos_storage[20];
    char * pos_ptr = &pos_storage[0];
    int i;
    address = 768;  /*default address */
    request_status();
    set_pointer(AXIS1_CMD);
    for(i=0;i<4;i++){   /* using auto increment feature of fast status */
      read_status(&word_high, &word_low, &fast_status);
      ltoa(fast_status, pos_ptr, 10);
      cputs(pos_ptr);   /* display axis 1, 2, 3,and 4 commanded positions */
      cputs("\r\n");
    }
    set_pointer(INO_STATUS);
    read_status(&word_high, &word_low, &fast_status);
    ino_bits = word_high;
  }
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Retrieving information from the fast status area using PASCAL: The following is a source code example written in Turbo PASCAL. This source code example is also provided on the DOS Support Disk in subdirectory SAMPLES, file TP5ØFAST.PAS.

```
Program testfast;

uses Crt;

const
  BASEP =        $0300;   { AT6nnn base port address }
  REQ_STATUS =   $48;     { AT6nnn fast status update }
  CS_UPDATED =   $08;     { AT6nnn card status update }
  FASTSTATUS =   2;       { fast status offset }
  STATUS =       4;       { set/clear status offset }
  AXIS1_CMD =    $00;     { pointer to axis 1 commanded position in fast
  status }

var
  i:integer;
  address:word;
  word_high, word_low:word;
  fast_status:real;
  pos_storage:string;

procedure request_status;
begin
  port[address+STATUS] := REQ_STATUS;  {request fast status update}
  {wait for fast status information to be updated}
  while((port[address+STATUS] and CS_UPDATED) = 0) do
    begin
    end;
end;

procedure set_pointer(status_offset:integer);
begin
  port[address+FASTSTATUS] := status_offset;
end;

procedure read_status(var status_high, status_low:word; var status:real);
begin
  status_high := portw[address+FASTSTATUS];
  status_low := portw[address+FASTSTATUS];
  status_high := (status_high shl 8) + (status_high shr 8);
  status_low := (status_low shl 8) + (status_low shr 8);
  status := status_high * 256.0 * 256.0 + status_low;
end;

begin
  address := BASEP;
  request_status;
  set_pointer(AXIS1_CMD);
  for i := 1 to 4 do
    begin
      read_status(word_high, word_low, fast_status);
      Writeln(fast_status:10:0);
    end;
end.
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Retrieving information from the fast status area using QuickBASIC: The following is a source code example written in QuickBASIC. This source code example is also provided on the DOS Support Disk in subdirectory SAMPLES, file QB45FAST.BAS.

```
DEFINT A-Z

DECLARE SUB requeststatus()
DECLARE SUB setpointer(statusoffset)
DECLARE SUB readstatus()
DECLARE SUB createposition()

'--- AT6nnn default address.
CONST BASEP = &H0300

CONST CSUPDATED   =   &H08    'AT6nnn card status update
CONST REQSTATUS   =   &H48    'tells AT6nnn that you want a status update
CONST AXIS1CMD    =   &H00    'Axis 1 commanded position in fast status area

'--- AT6nnn address offsets.
CONST FASTSTATUS  =   2       'offset to fast status area
CONST STATUS      =   4       'offset to set/clear status

'--- Globals
COMMON SHARED Address, Status4, Status3, Status2, Status1
COMMON SHARED Position#

'************************************************************************
'                              MAIN PROGRAM
'************************************************************************

address = BASEP
requeststatus
setpointer(AXIS1CMD)
readstatus
createposition
print "Axis 1 Commanded Position = ";position#

END


SUB requeststatus STATIC
'************************************************************************
'SUBPROGRAM : requeststatus
'************************************************************************
OUT address+STATUS, REQSTATUS
while (INP(address+STATUS) AND CSUPDATED) = 0
wend
END SUB


SUB setpointer(statusoffset) STATIC
'************************************************************************
'SUBPROGRAM : setpointer
'************************************************************************
out address+FASTSTATUS, statusoffset
END SUB


SUB readstatus STATIC
'************************************************************************
' SUBPROGRAM : readstatus
'************************************************************************
status4 = INP(address+FASTSTATUS)
status3 = INP(address+FASTSTATUS+1)
status2 = INP(address+FASTSTATUS)
status1 = INP(address+FASTSTATUS+1)
END SUB
```

Continued on next page ...

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

QuickBASIC Source Code Example *(continued)*

```
SUB createposition STATIC
'*************************************************************************
' SUBPROGRAM : createposition
' PURPOSE  : Manipulate the 4 bytes received from the fast status area into
'     a position value.  Only call createposition after calling readstatus.
'     This function should only be called when reading commanded position
'     or encoder position.
' REQUIRES : The 4 status bytes returned from readstatus - status1,
'     status2, status3, status4.
' RETURNS  : Position in global variable position#.
'*************************************************************************
IF (status4 AND NEGATIVEMASK) THEN 'If negative take the two's complement
 status4 = 255 - status4
 status3 = 255 - status3
 status2 = 255 - status2
 status1 = 255 - status1
 negvalue = 1
ELSE
 negvalue = 0
END IF

highword# = status4 * 256# + status3     'Create high word (upper 2 bytes)
lowword# = status2 * 256# + status1      'Create low word (lower 2 bytes)
position# = highword# * 256# * 256# + lowword#   'Create position (4 bytes)
IF (negvalue = 1) THEN
 position# = position# + 1           'Must add 1 for two's complement
 position# = 0 - position#           'Value should be negative
END IF

END SUB
```

Customizing the Fast Status Register

The number of customizable blocks differs between servo and a stepper 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 status 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.

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 the 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 (servos), refer to the table in the SSFR command description to determine the system update period (affected by the SSFR and INDAX). For the AT6n00 (steppers), one system update period is 2ms.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Option #	Information Provided *	Equivalent Status Command	Size (words)	Stepper (AT6n00)	Servo (AT6n50)
1	Commanded velocity (counts/sec)	TVEL	2 x 4	•	•
2	Axis status	TAS	2 x 4	•	•
3	Programmable input status (including triggers) Programmable output status (including aux. outputs) Limit status (hardware end-of-travel and home) Other input status (joystick and enable) A/D analog input voltage (joystick connector)	TIN TOUT **** TINO TANV	2 x 1 2 x 1 1 x 1 1 x 1 2 x 1	• • • • •	• • • • •
4	Interrupt status System status User status Time frame-mark (system update units – see SSFR) Timer value (milliseconds)	TINT TSS TUS n/a TIM	2 x 1 2 x 1 1 x 1 1 x 1 2 x 1	• • • • •	• • • • •
5	ANI input counts (ANI option only) **(counts, not volts)** ** Commanded DAC counts **(counts, not volts)** **	TANI TDAC	1 x 4 1 x 4		• •
6	Position error (counts)	TPER	2 x 4	•	•
7	Following status	TFS	2 x 4	***	•
8	Actual velocity (feedback device counts/sec)	TVELA	2 x 4		•
9	Captured commanded/motor position via trigger A (counts)	TPCCA/TPCMA	2 x 4	•	•
10	Captured commanded/motor position via trigger B (counts)	TPCCB/TPCMB	2 x 4	•	•
11	Captured commanded/motor position via trigger C (counts)	TPCCC/TPCMC	2 x 4	•	•
12	Captured commanded/motor position via trigger D (counts)	TPCCD/TPCMD	2 x 4	•	•
13	Captured actual position via trigger A (counts)	TPCEA	2 x 4	•	•
14	Captured actual position via trigger B (counts)	TPCEB	2 x 4	•	•
15	Captured actual position via trigger C (counts)	TPCEC	2 x 4	•	•
16	Captured actual position via trigger D (counts)	TPCED	2 x 4	•	•
17	Captured ANI value via trigger A (ADC counts)	TPCAA	2 x 4		•
18	Captured ANI value via trigger B (ADC counts)	TPCAB	2 x 4		•
19	Captured ANI value via trigger C (ADC counts)	TPCAC	2 x 4		•
20	Captured ANI value via trigger D (ADC counts)	TPCAD	2 x 4		•
21	Following master cycle position	TPMAS	2 x 4	***	•
22	Following master cycle number	TNMCY	2 x 4	***	•
23	Following net shift	TPSHF	2 x 4	***	•

* 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.

** ANI counts: 819 counts/volt; DAC counts: 2048 counts/10 volts.

*** Available for stepper product revisions 3.0 and higher.

**** Refer to the *Limits Bit Assignments* note on page 43.

Card Status and Interrupts to/from PC-AT (Base+4)

Status Read

D7	D6	D5	D4	D3	D2	D1	D0	Function
							X	Data is in controller's output buffer: 1 = data in buffer, 0 = buffer empty
						X		Controller ready to receive commands (256-byte buffer): 1 = buffer full, 0 = ready to receive
					X			Status of controller general purpose interrupt: 1 = interrupt exists, 0 = no interrupt
				X				Status of controller status area update interrupt to PC-AT: 1 = status updated, 0 = status not updated
			X					Status of controller master interrupt enable: 1 = interrupts enabled, 0 = interrupts disabled
		X						Kill has been requested, not yet executed: 1 = kill requested, 0 = kill not requested
	X							Controller operating system successfully loaded: 1 = not loaded, 0 = loaded
X								Bus controller successfully loaded: 1 = not loaded, 0 = loaded

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Status Write
(interrupt/status set/reset)

D7	D6	D5	D4	D3	D2	D1	D0	Function
1	0	0	0	0	0	0	1	Execute the requested kill.
0	1	0	0	0	0	1	0	Tell the controller to read data from the input buffer.
0	1	0	0	0	1	0	0	Clear the controller interrupt.
0	1	0	0	1	0	0	0	Request fast status update.

Status Write
(interrupt enable/disable)

D7	D6	D5	D4	D3	D2	D1	D0	Function
0	0	1					X	Interrupt when output buffer has data: 1 = enable interrupt, 0 = disable interrupt
0	0	1				X		Interrupt when input buffer empty : 1 = enable interrupt, 0 = disable interrupt
0	0	1			X			General controller to PC-AT interrupt: 1 = enable interrupt, 0 = disable interrupt
0	0	1		X				Card status updated interrupt: 1 = enable interrupt, 0 = disable interrupt
0	0	1	X					Master interrupt: 1 = enable interrupt, 0 = disable interrupt

Reading and Writing to the 6000 Controller

Source code routines are provided for the 6000 Series controller on the DOS Support Disk; these routines are proven and debugged. We strongly recommend that you use these drivers if your application is based in a higher level language. Four languages are supported—C (Microsoft 6.0 and Borland Turbo 2.0), BASIC (QuickBASIC 4.5), and PASCAL (Borland Turbo). However, we understand the need in some applications to know the data transfer protocol to and from the controller. Consequently, we have provided (below) a step-by-step process for communicating with the controller.

The controller transfers data in Word lengths to its Read/Write registers at Address+0 and Address+1 (the upper 8 bits of the word are at Address+0). The handshaking for the data transfers is performed at Address+4 (Status Register). Bits are described below from 0 - 7.

Sending Commands to the Controller:

1. Wait for bit #1 (second bit) to go low at Address+4 (256-byte buffer is empty).
2. Send the data to Address+0 and Address+1 one word at a time, ending the data block with a null character.
3. Set bits #1 and #6 at Address+4 (data waiting to be read).

Receiving Responses from the Controller:

1. Wait for bit #0 (first bit) to go high at Address+4 (output buffer has data).
2. Read a word of data from Address+0 and Address+1.
3. Repeat until bit #0 is low, or until 128 words are read.

DDE6000™ *(Dynamic Data Exchange server)*

TO ORDER
To order DDE6000, contact your local Automation Technology Center (ATC) or distributor.

☞ *Multiple 6000 products may be accessed simultaneously with the DDE6000.*

DDE6000 is a Dynamic Data Exchange (DDE) server that you can use to facilitate communication between a Windows application and your 6000 product family. For example, you might use DDE6000 with a third-part factory automation software and operator interface, such as Wonderware's In-Touch™. DDE6000 supports NetDDE, which allows operation over a Windows for Workgroups, Windows 95, or Windows NT network.

The DDE6000 *server*, a Windows program, provides access to 6000 controller data that can be useful to other Windows programs (DDE *clients*). DDE6000 supports three types of "conversations" with a DDE client:

- Cold Link ..... Allows a client to directly request a particular data item from DDE6000.
- Hot Link..... Allows a client to be automatically updated when a particular data item from the DDE6000 has changed.
- Warm Link.... Combination of cold link and hot link, where a client wants to be informed of changes in the DDE6000 data without immediately receiving the new data item.

For more information, refer to the DDE6000.HLP file on the DDE6000 diskette.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

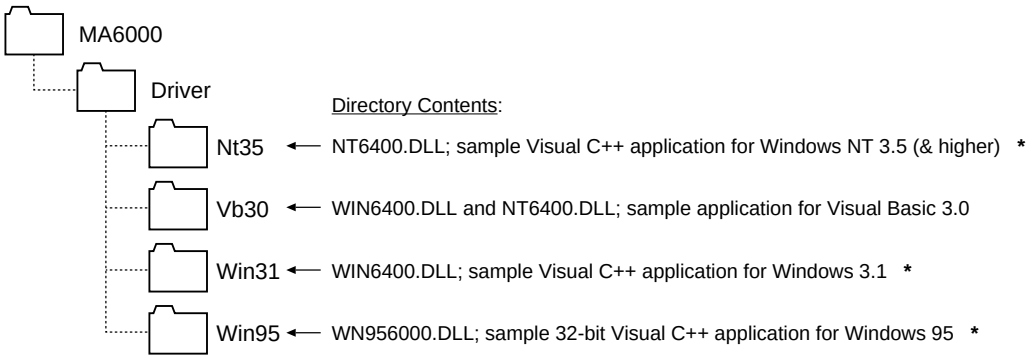
DLLs *(Dynamic Link Libraries)*

The information in this section is designed to help experienced Windows application programmers create Windows-based applications to interface with **bus-based 6000 Series controllers**.

To help you develop your own Windows applications, Compumotor provides dynamic link libraries (DLLs) for Windows 3.1, Windows 95, and Windows NT. The DLLs contain communication functions for use with all of Compumotor's bus-based 6000 Series control products; functions include sending commands to the controller, fetching responses from the controller, and polling status information from the controller's fast status area (*detailed function descriptions provided below*).

Windows 3.1 driver.....WIN6400.DLL
Windows 95 driver (32-bit).....WN956000.DLL
Windows NT driver.....NT6400.DLL

These DLLs are part of the Motion Architect installation options and are placed in your Motion Architect directory (default location is c:\MA6000\DRIVER). **NOTE**: If these directories do not appear in your Motion Architect directory, reinstall Motion Architect and be sure to select the desired support files from the "Custom Installation" dialog box during the installation process.



* README files are provided in these directories. They help instruct you on how to use the DLLs and the sample applications.

Instructions for using the DLLs with Visual Basic and Visual C++ are provided below.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Visual Basic™ Support

NOTE
To demonstrate how to use the DLL functions, we've provided all the files for a sample Visual Basic 3.0 project using WIN6400.DLL. Refer to page 57 for details.

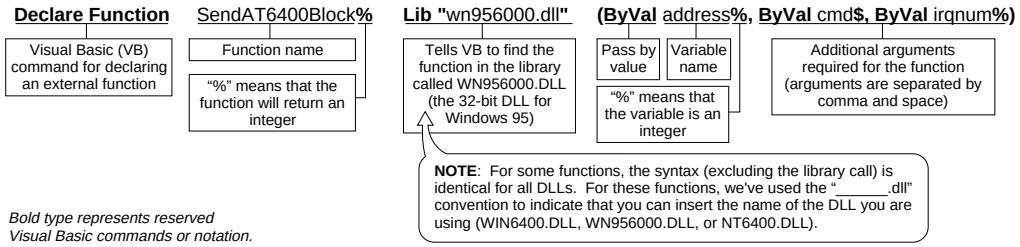
Visual Basic declarations for all DLL functions and subroutines are described below. Note that some functions are not applicable to all DLLs, and that the syntax for the **SendAT6400Block** and **RecvAT6400Block** functions is different when using WN956000.DLL.

TIP: Before you invoke Visual Basic, copy the desired DLL file (WIN6400.DLL, NT6400.DLL or WN956000.DLL) and your controller's _____.OPS (operating system) file from your Motion Architect directory to the Visual Basic root directory.

DLL Functions

SYNTAX EXPLANATION

(using the "SendAT6400Block" function as an example)



Bold type represents reserved Visual Basic commands or notation.

NT6400.DLL ONLY →

DLL Function: **SetNTParam**

Description: Initializes Windows NT driver for one card only. Call this function before any other communications with the 6000 product, including loading the operating system. *If you are using more than one 6000 card in your computer, use the* **SetNTMultiCardAddress** *function.*

Syntax: **Declare Function SetNTParam% Lib "nt6400.dll" (ByVal address%, ByVal irqnum%)**

"address" ..... Board address of the 6000 controller card (must be evenly divisible by 8).

"irqnum" ..... Set to zero (used internally by Motion Architect for interrupt support).

Return Value: Returns FALSE ("0") if function successful. Otherwise, TRUE (non-zero value).

NT6400.DLL ONLY →

DLL Function: **SetNTMultiCardAddress**

Description: Initializes Windows NT driver. Call this function before any other communications with the 6000 product, including loading the operating system. If you are using more than one 6000 card in your computer, call the **SetNTMultiCardAddress** function once for each card address.

Syntax: **Declare Function SetNTMultiCardAddress% Lib "nt6400.dll" (ByVal address%)**

"address" ..... Board address of the 6000 controller card (must be evenly divisible by 8).

Return Value: Returns FALSE ("0") if function successful. Otherwise, TRUE (non-zero value).

WN956000.DLL ONLY →

DLL Function: **SetDevice**

Description: Initializes Windows 95 driver. Call this function before any other communications with the 6000 product, including loading the operating system.

Syntax: **Declare Function SetDevice% Lib "wn956000.dll" (ByVal devnum$, ByVal address%)**

"devnum" ..... Device number assigned to the card. The first card addressed must be assigned device number zero (0); the next cards must be assigned consecutive numbers (i.e., 1,2,3, etc.).

"address" ..... Board address of the 6000 controller card (must be evenly divisible by 8).

Return Value: Returns FALSE ("0") if function successful. Otherwise, TRUE (non-zero value).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

DLL Functions for Visual Basic
(continued)

DLL Function:	**SendAT6400Block**
Description:	Send a block of up to 256 characters to a 6000 bus-based product. **SendAT6400Block** will wait until the 6000 controller is ready before sending. By default SendAT6400 will wait forever until the 6000 controller is ready. **SetTimeout** allows control over this waiting period.
Syntax:	For WIN6400.DLL and NT6000.DLL: **Declare Function SendAT6400Block% Lib "_____.dll" (ByVal address%, ByVal cmd$, ByVal irqnum%)** For WN956000.DLL: **Declare Function SendAT6400Block% Lib "wn956000.dll" (ByVal address%, ByVal cmd$)** "address".....Board address of the 6000 controller card (must be evenly divisible by 8). The default address on the 6000 controller (selected with DIP switches) is 768; to change the address, refer to the product's *Installation Guide*. "cmd".....Pointer to a null-terminated string that contains one or more 6000 series commands. Commands must be separated with a carriage return or colon. The buffer that *cmd* points to should be 258 bytes in size. "irqnum".....Set to zero (used internally by Motion Architect for interrupt support).
Return Value:	"-1" indicates that the operating system is not loaded. "-2" indicates that the function has timed out (see **SetTimeout**). **SendAT6400Block** also returns the number of bytes sent to the 6000 product.

DLL Function:	**RecvAT6400Block**
Description:	Poll the controller for any response it may have in its output buffer. Fetches a block of up to 256 characters (block is terminated with a null).
Syntax:	For WIN6400.DLL and NT6000.DLL: **Declare Function RecvAT6400Block% Lib "_____.dll" (ByVal address%, ByVal resp$, ByVal irqnum%)** For WN956000.DLL: **Declare Function RecvAT6400Block% Lib "wn956000.dll" (ByVal address%, ByVal resp$)** "address".....Board address of the 6000 controller card (must be evenly divisible by 8). "resp".....Pointer to a 258 byte response buffer. "irqnum".....Set to zero (used internally by Motion Architect for interrupt support).
Return Value:	"-1" indicates that the operating system is not loaded. "-2" indicates that the function has timed out (see **SetTimeout**). **RecvAT6400Block** also returns the number of bytes received from the 6000 product.

DLL Function:	**osload**
Description:	Download the operating system to the controller.
Syntax:	**Declare Function osload% Lib "_____.dll" (ByVal address%, ByVal options$, ByVal handle%)** "address".....Board address of the 6000 controller card (must be evenly divisible by 8). "options".....Pointer to a string of operating system loader options: • "at6400.ops" downloads AT6n00 operating system (same operating system for the AT6200, AT6400, OEM-AT6200, and OEM-AT6400) • "at6250.ops" downloads AT6250 operating system • "at6450.ops" downloads AT6450 operating system • "/port=768" allows operating system to be sent to a particular address • "/quiet" keeps **osload** form displaying the meter dialog and error messages "handle".....Parent window handle for **osload**'s meter dialog and message boxes. For no parent, set to zero.
Return Value:	"0" indicates that the operating system downloaded successfully. Value greater than 0 for error conditions (see enum osload_errors in WIN6400.H).

DLL Function:	**IsOSLoaded**
Description:	Determine if the operating system has been loaded to the 6000 controller.
Syntax:	**Declare Function IsOSLoaded% Lib "_____.dll" (ByVal address%)** "address".....Board address of the 6000 controller card (must be evenly divisible by 8).
Return Value:	Returns TRUE (non-zero value) if the 6000 operating system has been loaded. Otherwise, returns FALSE ("0").

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

DLL Function:	**request_status**
Description:	Tell the controller to update the information in its fast status area. This function will wait until the fast status area has been updated. See **set_pointer**.
Syntax:	**Declare Function request_status% Lib "_____.dll" (ByVal address%)** "address" ..... Board address of the 6000 controller card (must be evenly divisible by 8).
Return Value:	"-1" indicates that the operating system is not loaded. "-2" indicates that the function has timed out (see **SetTimeout**).

DLL Function:	**set_pointer**
Description:	Set a pointer to the data to be retrieved from the fast status area. See page 56 for fast status structure.
Syntax:	**Declare Sub set_pointer Lib "_____.dll" (ByVal address%, ByVal status_offset%)** "address" .....Board address of the 6000 controller (must be evenly divisible by 8). "status_offset".....Offset into fast status area.
Return Value:	NONE

DLL Function:	**read_status**
Description:	Fetch a fast status data item (4 bytes) pointed to by **set_pointer**. See **set_pointer**.
Syntax:	**Declare Sub read_status Lib "_____.dll" (ByVal address%, status_high%, status_low%, status&)** "address" .....Board address of the 6000 controller card (must be evenly divisible by 8). "status_high".....Pointer to word that will contain the high word of the retrieved fast status data. "status_low".....Pointer to word that will contain the low word of the retrieved fast status data. "status".....Pointer to a double word that will contain the retrieved fast status data item.
Return Value:	NONE

DLL Function:	**IsAT6400Ready**
Description:	Check to see if the controller's input buffer (256 bytes) is empty, thereby being ready to receive another command.
Syntax:	**Declare Function IsAT6400Ready% Lib "_____.dll" (ByVal address%)** "address" ..... Board address of the 6000 controller card (must be evenly divisible by 8).
Return Value:	Returns TRUE (non-zero value) if 6000 input buffer is empty. Otherwise, returns FALSE ("0").

DLL Function:	**SetTimeout**
Description:	Set the timeout value for **SendAT6400Block**, **GetFastStatus**, and **GetExFastStatus**, and **Request_Status**.
Syntax:	**Declare Sub SetTimeout Lib "_____.dll" (ByVal timeout&)** "timeout" ..... Timeout value in milliseconds. A value of zero sets the timeout to infinity.
Return Value:	NONE

DLL Function:	**Delay**
Description:	Time delay in milliseconds.
Syntax:	For WN956000.DLL and NT6000.DLL: **Declare Sub Delay Lib "_____.dll" (ByVal timedelay%)** For WIN6400.DLL: **Declare Sub Delay Lib "_____.dll" (ByVal timedelay&)** "timedelay"..... Time delay in milliseconds.
Return Value:	NONE

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

DLL Function:	**SendAT6400File**
Description:	Downloads a file of 6000 Series commands to a 6000 series product (one line at a time). Before downloading the file, ERRDEF and ERR0K are set to zero. After downloading the file, ERRDEF and ERR0K are set to their default values.
Syntax:	**Declare Function SendAT6400File% Lib "_____.dll" (ByVal handle%, ByVal address%, ByVal irqnum%, ByVal filename$, ByVal options$)** "handle".....Parent window handle for **SendAT6400File**'s message boxes. For no parent, set to zero. "address".....Board address of the 6000 controller card (must be evenly divisible by 8). "irqnum".....Set to zero (used internally by Motion Architect for interrupt support). "filename".....Pointer to filename of file that is to be sent to the 6000 product. "options".....Pointer to options string. Set to null string ("") to get the message boxes. Set to "/quiet" to keep **SendAT6400File** from displaying the hour glass cursor and message boxes.
Return Value:	"-1" indicates that the operating system is not loaded. "-2" indicates that the function has timed out (see **SetTimeout**). "-3" indicates that the file ("filename") could not be opened. **SendAT6400File** also returns the number of bytes sent to the 6000 product.

DLL Function:	**GetFastStatus**
Description:	Fetches Blocks 1 through 6 of the 6000 fast status area. See page 56 for fast status structure.
Syntax:	**Declare Function GetFastStatus% Lib "_____.dll" (ByVal address%, faststatus As AT6400INFO)** "address".....Board address of the 6000 controller card (must be evenly divisible by 8). "faststatus"...Fast status structure. See page 56 for fast status structure (defined as "AT6400INFO").
Return Value:	"1" indicates no errors. "-1" indicates that the operating system is not loaded. "-2" indicates that the function has timed out (see **SetTimeout**). **GetFastStatus** also returns the number of bytes sent to the 6000 product.

DLL Function:	**GetExFastStatus**
Description:	**SERVOS ONLY** Fetches Blocks 7 and 8 of the 6000 fast status area.
Syntax:	**Declare Function GetExFastStatus% Lib "_____.dll" (ByVal address%, ByVal block7%, ByVal block8%, faststatus As AT6400INFO)** "address".....Board address of the 6000 controller card (must be evenly divisible by 8). "block7".....Integer code representing data type in Block 7 of the fast status area (see enum exfaststatus in WIN6400.H). "block8".....Integer code representing data type in Block 8 of the fast status area (see enum exfaststatus in WIN6400.H). "faststatus"...Fast status structure. See page 56 for fast status structure (defined as "AT6400INFO").
Return Value:	"1" indicates no errors. "-1" indicates that the operating system is not loaded. "-2" indicates that the function has timed out (see **SetTimeout**). **GetExFastStatus** also returns the number of bytes sent to the 6000 product.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Sample file for fast status structure, function declarations, and global variables (WIN6400.DLL):

SEE ALSO: Driver\vb30\AT6400.BAS

```
'------------------------------------------------------------------------
' Get faststatus structure -- also provided in DRIVER\VB30\AT6400.BAS
'------------------------------------------------------------------------
Type AT6400INFO
    MotorPos(1 To 4) As Long           ' commanded position (counts)
    EncPos(1 To 4) As Long             ' actual position (counts)
    MotorVel(1 To 4) As Long           ' commanded velocity (counts/sec)
    AxisStatus(1 To 4) As Long         ' axis status (TAS)
    IntStatus As Long                  ' interrupt status (TINT)
    SysStatus As Long                  ' system status (TSS)
    UserStatus As Integer              ' user status (TUS)
    Timer As Long                      ' timer value (TIM - milliseconds)
    Counter As Integer                 ' time frame counter (2ms per count)
    ProgIn As Long                     ' programmable input status (TIN)
    ProgOut As Long                    ' programmable output status (TOUT)
    Limits As Integer                  ' limit status (TLIM)
    Other As Integer                   ' other input status (TINO)
    Analog As Long                     ' lo-res analog input voltage (TANV)
    PosOffset(1 To 4) As Long          ' position offset (68000-DSP)
    EncVel(1 To 4) As Long             ' actual velocity (counts/sec)
    XEncPos(1 To 1) As Long            ' extra encoder position (counts)
    ANI(1 To 4) As Integer             ' hi-res analog input voltage (TANI)
    ANIOffset(1 To 4) As Long          ' hi-res ANI offset
    DAC(1 To 4) As Integer             ' commanded DAC count (TDAC)
    PosError(1 To 4) As Long           ' position error (TPER - counts)
    MasterCycleNum(1 To 4) As Long     ' following master cycle number (TNMCY)
    MasterCyclePos(1 To 4) As Long     ' following master cycle position (TPMAS)
    FollStatus(1 To 4) As Long         ' following status (TFS)
    PosShift(1 To 4)   As Long         ' following net shift (TPSHF)
    MasterVel(1 To 4) As Long          ' following master velocity (TVMAS)
    TPCCA(1 To 4) As Long              ' captured commanded position via trigger A (counts)
    TPCCB(1 To 4) As Long              ' captured commanded position via trigger B (counts)
    TPCCC(1 To 4) As Long              ' captured commanded position via trigger C (counts)
    TPCCD(1 To 4) As Long              ' captured commanded position via trigger D (counts)
    TPCEA(4) As Long                   ' captured actual position via trigger A (counts)
    TPCEB(1 To 4) As Long              ' captured actual position via trigger B (counts)
    TPCEC(1 To 4) As Long              ' captured actual position via trigger C (counts)
    TPCED(1 To 4) As Long              ' captured actual position via trigger D (counts)
    TPCAA(1 To 4) As Long              ' captured ANI value via trigger A (counts)
    TPCAB(1 To 4) As Long              ' captured ANI value via trigger B (counts)
    TPCAC(1 To 4) As Long              ' captured ANI value via trigger C (counts)
    TPCAD(1 To 4) As Long              ' captured ANI value via trigger D (counts)
    dfVAR11 As Long                    ' variable VAR11
    dfVAR12 As Long                    ' variable VAR12
    dfVAR13 As Long                    ' variable VAR13
    dfVAR14 As Long                    ' variable VAR14
End Type

'------------------------------------------------------------------------
' win6400.dll FUNCTION DECLARATIONS
'------------------------------------------------------------------------
Declare Function SendAT6400Block% Lib "win6400.dll" (ByVal address%, ByVal cmd$, ByVal irqnum%)
Declare Function RecvAT6400Block% Lib "win6400.dll" (ByVal address%, ByVal resp$, ByVal irqnum%)
Declare Function osload% Lib "win6400.dll" (ByVal address%, ByVal options$, ByVal handle%)
Declare Function IsOSLoaded% Lib "win6400.dll" (ByVal address%)
Declare Function request_status% Lib "win6400.dll" (ByVal address%)
Declare Sub set_pointer Lib "win6400.dll" (ByVal address%, ByVal status_offset%)
Declare Sub read_status Lib "win6400.dll" (ByVal address%, status_high%, status_low%, status&)
Declare Function IsAT6400Ready% Lib "win6400.dll" (ByVal address%)
Declare Sub SetTimeout Lib "win6400.dll" (ByVal timeout&)
Declare Sub Delay Lib "win6400.dll" (ByVal timedelay&)
Declare Function SendAT6400File% Lib "win6400.dll" (ByVal handle%, ByVal address%, ByVal irqnum%,
ByVal filename$, ByVal options$)
Declare Function SetNTParam% Lib "win6400.dll" (ByVal address%, ByVal irqnum%)
Declare Function GetFastStatus% Lib "win6400.dll" (ByVal address%, faststatus As AT6400INFO)
Declare Function GetExFastStatus% Lib "win6400.dll" (ByVal address%, ByVal block7%, ByVal block8%,
faststatus As AT6400INFO)

'------------------------------------------------------------------------
' mmsystem.dll FUNCTION DECLARATION
'------------------------------------------------------------------------
Declare Function TimeGetTime& Lib "mmsystem.dll" ()

'------------------------------------------------------------------------
' Global Variables Used
'------------------------------------------------------------------------
Global response As String * 258      'response for the RecvAt6400Block
Global address As Integer            'device address for the at card
Global faststatus As AT6400INFO      'structure holding at6400 info
Global cmd$                           'command string to send in SendAt6400Block
```

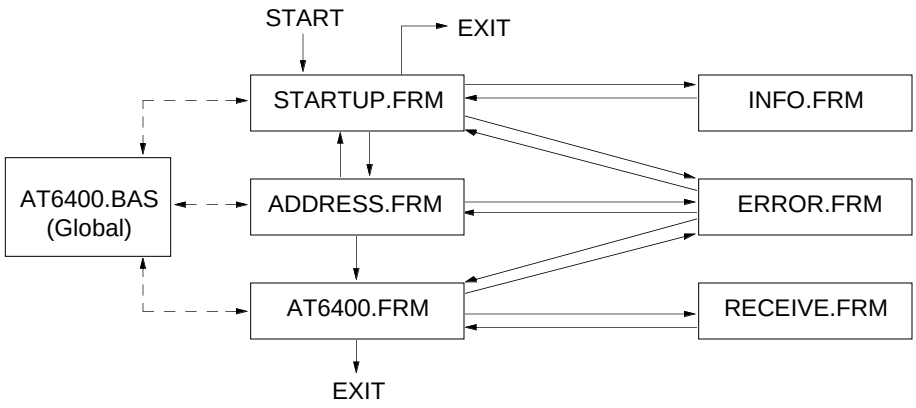

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

VB 3.0 Application Example

To demonstrate how to use the DLL functions, we've provided all the files for a sample Visual Basic *project* using WIN6400.DLL for VB3.0 (see table below for list of files) The files are located in the MA6000\DRIVER\VB30 directory. The project was designed to be executed from Visual Basic 3.0 with no modifications.

Project File (*.MAK**)**	**Forms (***.FRM**)**	**Basic Modules (***.BAS**)**	**Icons (***.ICO**)**
AT6400.MAK	ADDRESS.FRM	AT6400.BAS	ADDRESS.ICO
	AT6400.FRM		AT6400.ICO
	AT_ERROR.FRM		BLKLIMIT.ICO
	INFO.FRM		ERROR.ICO
	RECEIVE.FRM		NO_LIMIT.ICO
	STARTUP.FRM		REDLIMIT.ICO
			TSTPANEL.ICO

To initiate the application, invoke Visual Basic and open the AT6400.MAK file by choosing **Open Project** from the **File** menu. Then select **Start** from the **Run** menu to initiate the application. The application follows the structure illustrated below.



The global module, AT6400.BAS, contains code necessary to declare the functions and subroutines contained within WIN6400.DLL. The declarations for those functions and subroutines are provided earlier in this section.

Subsequent calls to these declared functions and subroutines allow for communication between the Visual Basic application program and the controller. Examples of each function and subroutine are provided within the "forms" of the project. Refer to the form next to the function or subroutine in the table below for specific calling information.

Function or Subroutine	**Form**	**Object**	**Event Procedure**
SendAT6400Block	STARTUP.FRM	BTN_Yes	Click
RecvAT6400Block	AT6400.FRM	MNU_Rev_Update	Click
OSLoad	STARTUP.FRM	BTN_Yes	Click
IsOSLoad	STARTUP.FRM	BTN_No	Click
Request_Status	AT6400.FRM	TIM_Update	Timer
Set_Pointer	AT6400.FRM	TIM_Update	Timer
Read_Status	AT6400.FRM	TIM_Update	Timer

The starting point for the application is the form STARTUP.FRM. This is established by choosing **Select Startup Form** under the **Run** menu.

For a complete listing of the Visual Basic application described above, open the project file (AT6400.MAK) by choosing **Open Project** from the **File** menu. Then select **Print** from the **File** menu, and print out all forms and code.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Visual C++ declarations for all DLL functions and subroutines are described below. Note that some functions are not applicable to all DLLs, and that the "irqnum" variable is not part of the syntax for the **SendAT6400Block** and **RecvAT6400Block** function when using WN956000.DLL.

TIP: Before you invoke Visual C++, copy the desired DLL file (WIN6400.DLL, NT6400.DLL or WN956000.DLL) and your controller's _____.OPS (operating system) file from your Motion Architect directory to the Visual C++ root directory.

SAMPLE APPLICATIONS: Within each DLL's subdirectory are source files for sample applications that exploit the DLL's functions. Refer to page 62 for details.

DLL Functions

Refer to the Header (.H) file

The primary purpose of this section is to identify the sequence and purpose of the variables used in each DLL function. The function declarations below are also provided in the respective header file (WIN6400.H, WN956000.H, or NT6400.H). Refer also to the header file for the structure of the fast status area.

NT6400.DLL ONLY →

DLL Function:	**SetNTParam**
Description:	Initializes Windows NT driver for one card only. Call this function before any other communications with the 6000 product, including loading the operating system. *If you are using more than one 6000 card in your computer, use the* **SetNTMultiCardAddress** *function.*
Syntax:	nt6400.dll ..... **BOOL WINAPI SetNTParam(DWORD address, DWORD irqnum)**
Variables:	"address"..... Board address of the 6000 controller card (must be evenly divisible by 8).
	"irqnum"..... Set to zero (used internally by Motion Architect for interrupt support).
Return Value:	Returns FALSE ("0") if function successful. Otherwise, TRUE (non-zero value).

NT6400.DLL ONLY →

DLL Function:	**SetNTMultiCardAddress**
Description:	Initializes Windows NT driver. Call this function before any other communications with the 6000 product, including loading the operating system. If you are using more than one 6000 card in your computer, call the **SetNTMultiCardAddress** function once for each card address.
Syntax:	nt6400.dll ..... **BOOL WINAPI SetNTMultiCardAddress(DWORD address)**
Variables:	"address"..... Board address of the 6000 controller card (must be evenly divisible by 8).
Return Value:	Returns FALSE ("0") if function successful. Otherwise, TRUE (non-zero value).

WN956000.DLL ONLY →

DLL Function:	**SetDevice**
Description:	Initializes Windows 95 driver. Call this function before any other communications with the 6000 product, including loading the operating system.
Syntax:	wn956000.dll .... **BOOLEAN WINAPI SetDevice(long devnum, short address)**
Variables:	"devnum"..... Device number assigned to the card. The first card addressed must be assigned device number zero (0); the next cards must be assigned consecutive numbers (i.e., 1,2,3, etc.).
	"address"..... Board address of the 6000 controller card (must be evenly divisible by 8).
Return Value:	Returns FALSE ("0") if function successful. Otherwise, TRUE (non-zero value).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

DLL Functions for Visual C++
(continued)

DLL Function:	**SendAT6400Block**
Description:	Send a block of up to 256 characters to a 6000 bus-based product. **SendAT6400Block** will wait until the 6000 controller is ready before sending. By default SendAT6400 will wait forever until the 6000 controller is ready. **SetTimeout** allows control over this waiting period.
Syntax:	wn956000.dll.....**short WINAPI SendAT6400Block(short address, LPSTR cmd)** nt6400.dll.....**short WINAPI SendAT6400Block(short address, LPSTR cmd, short irqnum)** win6400.dll.....**int FAR PASCAL SendAT6400Block(WORD address, LPSTR cmd, int irqnum)**
Variables:	"address".....Board address of the 6000 controller card (must be evenly divisible by 8). The default address on the 6000 controller (selected with DIP switches) is 768; to change the address, refer to the product's *Installation Guide*. "cmd".....Pointer to a null-terminated string that contains one or more 6000 series commands. Commands must be separated with a carriage return or colon. The buffer that *cmd* points to should be 258 bytes in size. "irqnum".....Set to zero (used internally by Motion Architect for interrupt support).
Return Value:	"-1" indicates that the operating system is not loaded. "-2" indicates that the function has timed out (see **SetTimeout**). **SendAT6400Block** also returns the number of bytes sent to the 6000 product.

DLL Function:	**RecvAT6400Block**
Description:	Poll the controller for any response it may have in its output buffer. Fetches a block of up to 256 characters (block is terminated with a null).
Syntax:	wn956000.dll.....**short WINAPI RecvAT6400Block(short address, LPSTR resp)** nt6400.dll.....**short WINAPI RecvAT6400Block(short address, LPSTR resp, short irqnum)** win6400.dll.....**int FAR PASCAL RecvAT6400Block(WORD address, LPSTR resp, int irqnum)**
Variables:	"address".....Board address of the 6000 controller card (must be evenly divisible by 8). "resp".....Pointer to a 258 byte response buffer. "irqnum".....Set to zero (used internally by Motion Architect for interrupt support).
Return Value:	"-1" indicates that the operating system is not loaded. "-2" indicates that the function has timed out (see **SetTimeout**). **RecvAT6400Block** also returns the number of bytes received from the 6000 product.

DLL Function:	**osload**
Description:	Download the operating system to the controller.
Syntax:	wn956000.dll & nt6400.dll.....**short WINAPI osload(unsigned short address, LPSTR options, HWND handle)** win6400.dll.....**int FAR PASCAL osload(WORD address, LPSTR options, HWND handle)**
Variables:	"address".....Board address of the 6000 controller card (must be evenly divisible by 8). "options".....Pointer to a string of operating system loader options: • "at6400.ops" downloads AT6n00 operating system (same operating system for the AT6200, AT6400, OEM-AT6200, and OEM-AT6400) • "at6250.ops" downloads AT6250 operating system • "at6450.ops" downloads AT6450 operating system • "/port=768" allows operating system to be sent to a particular address • "/quiet" keeps **osload** form displaying the meter dialog and error messages "handle".....Parent window handle for **osload**'s meter dialog and message boxes. For no parent, set to zero.
Return Value:	"0" indicates that the operating system downloaded successfully. Value greater than 0 for error conditions (see enum osload_errors in WIN6400.H).

DLL Function:	**IsOSLoaded**
Description:	Determine if the operating system has been loaded to the 6000 controller.
Syntax:	wn956000.dll & nt6400.dll.....**short WINAPI IsOSLoaded(short address)** win6400.dll.....**int FAR PASCAL IsOSLoaded(WORD address)**
Variable:	"address".....Board address of the 6000 controller card (must be evenly divisible by 8).
Return Value:	Returns TRUE (non-zero value) if the 6000 operating system has been loaded. Otherwise, returns FALSE ("0").

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

DLL Functions for Visual C++
(continued)

DLL Function:	**request_status**
Description:	Tell the controller to update the information in its fast status area. This function will wait until the fast status area has been updated. See **set_pointer**.
Syntax:	wn956000.dll & nt6400.dll.....**short WINAPI request_status(short address)** win6400.dll .....**int FAR PASCAL request_status(WORD address)**
Variable:	"address"..... Board address of the 6000 controller card (must be evenly divisible by 8).
Return Value:	"-1" indicates that the operating system is not loaded. "-2" indicates that the function has timed out (see **SetTimeout**).

DLL Function:	**set_pointer**
Description:	Set a pointer to the data to be retrieved from the fast status area. See header file (WN956000.H, NT6400.H, or WIN6400.H) for fast status structure.
Syntax:	wn956000.dll & nt6400.dll ..... **void WINAPI set_pointer(short address, short status_offset)** win6400.dll ..... **void FAR PASCAL set_pointer(WORD address, WORD status_offset)**
Variables:	"address"..... Board address of the 6000 controller (must be evenly divisible by 8). "status_offset" .. Offset into fast status area. See header file for fast status structure.
Return Value:	NONE

Example	`// request fast status update`
	`request_status(address)`
	`// point to  block 5 of fast status area`
	`set_pointer(address,  INPUT_STATUS);`
	`// fetch TIN status`
	`read_status(address,  &word_high,  &word_low,  &fast_status);`
	`dwProgIn = fast_status;`
	`// fetch TOUT status`
	`read_status(address,  &word_high,  &word_low,  &fast_status);`
	`dwProgOut = fast_status;`
	`// fetch TLIM and TINO status`
	`read_status(address,  &word_high,  &word_low,  &fast_status);`
	`wLimits = word_high;`
	`wOther = word_low;`
	`// fetch TANV status`
	`read_status(address,  &word_high,  &word_low,  &fast_status);`
	`dwAnalog = fast_status;`

DLL Function:	**read_status**
Description:	Fetch a fast status data item (4 bytes) pointed to by **set_pointer**. See **set_pointer**.
Syntax:	wn956000.dll & nt6400.dll ..... **void WINAPI read_status (short address, LPWORD status_high, LPWORD status_low, LPDWORD status)** win6400.dll ..... **void FAR PASCAL read_status (WORD address, LPWORD status_high, LPWORD status_low, LPDWORD status)**
Variables:	"address"..... Board address of the 6000 controller card (must be evenly divisible by 8). "status_high" .... Pointer to word that will contain the high word of the retrieved fast status data. "status_low" ..... Pointer to word that will contain the low word of the retrieved fast status data. "status" ..... Pointer to a double word that will contain the retrieved fast status data item.
Return Value:	NONE

DLL Function:	**IsAT6400Ready**
Description:	Check to see if the controller's input buffer (256 bytes) is empty, thereby being ready to receive another command.
Syntax:	wn956000.dll & nt6400.dll.....**BOOL WINAPI IsAT6400Ready(short address)** win6400.dll .....**BOOL FAR PASCAL IsAT6400Ready(WORD address)**
Variable:	"address"..... Board address of the 6000 controller card (must be evenly divisible by 8).
Return Value:	Returns TRUE (non-zero value) if 6000 input buffer is empty. Otherwise, returns FALSE ("0").

DLL Function:	**SetTimeout**
Description:	Set the timeout value for **SendAT6400Block**, **GetFastStatus**, and **GetExFastStatus**, and **Request_Status**.
Syntax:	wn956000.dll & nt6400.dll.....**void WINAPI SetTimeout (DWORD timeout)** win6400.dll .....**void FAR PASCAL SetTimeout (DWORD timeout)**
Variable:	"timeout"..... Timeout value in milliseconds. A value of zero sets the timeout to infinity.
Return Value:	NONE

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

DLL Functions for Visual C++
(continued)

DLL Function:	**Delay**
Description:	Time delay in milliseconds.
Syntax:	wn956000.dll & nt6400.dll.....**void WINAPI Delay (int delay)** win6400.dll.....**void FAR PASCAL Delay (DWORD delay)**
Variable:	"timedelay".....Time delay in milliseconds.
Return Value:	NONE
DLL Function:	**SendAT6400File**
Description:	Downloads a file of 6000 Series commands to a 6000 series product (one line at a time). Before downloading the file, ERRDEF and ERR0K are set to zero. After downloading the file, ERRDEF and ERR0K are set to their default values.
Syntax:	wn956000.dll & nt6400.dll.....**short WINAPI SendAT6400File(HWND handle, short address, short irqnum, LPSTR filename, LPSTR options)** win6400.dll.....**int FAR PASCAL SendAT6400File(HWND handle, WORD address, int irqnum, LPSTR filename, LPSTR options)**
Variables:	"handle".....Parent window handle for **SendAT6400File**'s message boxes. For no parent, set to zero. "address".....Board address of the 6000 controller card (must be evenly divisible by 8). "irqnum".....Set to zero (used internally by Motion Architect for interrupt support). "filename".....Pointer to filename of file that is to be sent to the 6000 product. "options".....Pointer to options string. Set to null string ("") to get the message boxes. Set to "/quiet" to keep **SendAT6400File** from displaying the hour glass cursor and message boxes.
Return Value:	"-1" indicates that the operating system is not loaded. "-2" indicates that the function has timed out (see **SetTimeout**). "-3" indicates that the file ("filename") could not be opened. **SendAT6400File** also returns the number of bytes sent to the 6000 product.

DLL Function:	**GetFastStatus**
Description:	Programs and fetches Blocks 1 through 6 of the 6000 fast status area. See header file (WN956000.H, NT6400.H, or WIN6400.H) for fast status structure.
Syntax:	wn956000.dll & nt6400.dll.....**short WINAPI GetFastStatus(WORD address, LPAT6400INFO lpAT6400Info)** win6400.dll.....**int FAR PASCAL GetFastStatus(WORD address, LPAT6400INFO lpAT6400Info)**
Variables:	"address"..... Board address of the 6000 controller card (must be evenly divisible by 8). "lpAT6400Info"..... Fast status structure. See header file (WN956000.H, NT6400.H, or WIN6400.H) for fast status structure – defined as "AT6400INFO".
Return Value:	"1" indicates no errors. "-1" indicates that the operating system is not loaded. "-2" indicates that the function has timed out (see **SetTimeout**). **GetFastStatus** also returns the number of bytes sent to the 6000 product.

DLL Function:	**GetExFastStatus**
Description:	**SERVOS ONLY:** Programs and fetches Block 7 and Block 8 of the 6000 fast status area.
Syntax:	wn956000.dll & nt6400.dll.....**short WINAPI GetExFastStatus(WORD address, short block7, short block8, LPAT6400INFO lpAT6400Info)** win6400.dll.....**int FAR PASCAL GetExFastStatus(WORD address, int block7, int block8, LPAT6400INFO lpAT6400Info)**
Variables:	"address"..... Board address of the 6000 controller card (must be evenly divisible by 8). "block7"..... Integer code representing data type in Block 7 of the fast status area (see enum exfaststatus in header file). "block8"..... Integer code representing data type in Block 8 of the fast status area (see enum exfaststatus in header file). "lpAT6400Info"..... Fast status structure. See header file (WN956000.H, NT6400.H, or WIN6400.H) for fast status structure – defined as "AT6400INFO".
Return Value:	"1" indicates no errors. "-1" indicates that the operating system is not loaded. "-2" indicates that the function has timed out (see **SetTimeout**). **GetExFastStatus** also returns the number of bytes sent to the 6000 product.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Visual C++ Sample Applications

Source code for sample Windows applications are provided for each DLL. (For additional information, refer to the README.TXT files in the respective sub-directory.)

- WN956000.DLL
  - Directory is MA6000\Drivers\win95
  - Windows 95 (32-bit) application is called “Mawin95”
  - For details about each source file, refer to the README.TXT file.
  - The application must be compiled using the 32-bit version of Visual C++.

- NT6400.DLL
  - Directory is MA6000\Drivers\nt35
  - Windows NT 3.5 application is called “Motarcnt”
  - For details about each source file, refer to the README.TXT file.
  - The application must be compiled using the 32-bit version of Visual C++.

- WIN6400.DLL
  - Directory is MA6000\Drivers\win31
  - Windows 3.1 application is called “Oppanel”
  - The application is an operator panel that allows you to start and stop motion on each axis. Also, it continuously displays motion and limit status.
  - You can compile the Oppanel application using the Microsoft C compiler (6.0 and above) and the Microsoft Windows SDK.
  - Source files:

Make File.....	OPPANEL.MAK
Code Files.....	OPPANEL.C OPPANEL.H OPPANEL.HH WIN6400.H
Resource Files.....	OPPANEL.RC OPPANEL.DLG OPPANEL.ICO EOTLIMIT.ICO HOMLIMIT.ICO NOLIMIT.ICO
Link Files	OPPANEL.LNK OPPANEL.DEF
Library Files.....	WIN6400.DLL WIN6400.LIB METER.DLL * NT6400.DLL
Executable File....	OPPANEL.EXE

* METER.DLL provides a meter dialog for the 6000 operating system download process via **osload**. If this DLL is not present, an hourglass will appear during the download of the 6000 operating system.

Motion OCX Toolkit™ *(bus-based products only)*

TO ORDER

To order the Motion OCX Toolkit, contact your local Automation Technology Center (ATC) or distributor.

The Motion OCX Toolkit provides 32-bit Ole Custom Controls (OCXs) designed to run under Windows 95 or Windows NT. Controls include:

- Communications Shell — control basic communication with the 6000 product, including interrupt handling and sending/receiving files.
- Fast-Status Polling — poll the 6000 product's fast status register (see page 43 for description of fast status area).
- Terminal — terminal emulator.

The OCX controls can be used with Visual Basic 4.0, Delphi 2.0. Visual C++ 4.x, or any 32 bit development environment that can contain OCX controls.

For more information, refer to the ***Motion OCX Toolkit User Guide***.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

PC-AT Interrupts

NOTE

This section uses a generic reference ("AT6nnn") to represent all 6000 Series bus-based products. When referring to the file names and programming examples, substitute the name of your product where you read "AT6nnn". (e.g., if you are using the AT6450, type "AT6450")

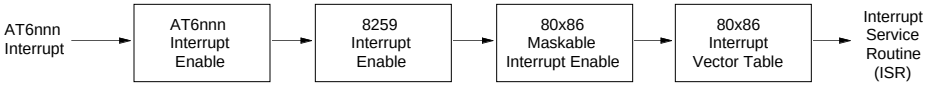
Exceptions: For the OEM-AT6400, AT6200, & OEM-AT6200 products, type "AT6400".

This section describes how to write PC-AT software that exploits the interrupt capability of the bus-based 6000 Series controller (AT6nnn). To best understand the interrupt function, this section is organized as follows:

- AT6nnn interrupt path
- How to use interrupts
- Practical Example: Using the interrupt-driven terminal emulator (`MC6ØTRMI`) provided in the DOS Support Disk

AT6nnn Interrupt Path

The 6000 Series controller is capable of interrupting the PC-AT. When interrupted, the 80x86 processor executes an interrupt service routine (ISR). The path that the controller interrupt takes to get to the 80x86 processor is shown below.



Notice that the AT6nnn interrupt must get by three different interrupt enables before the 80x86 processor can be interrupted:

1. Interrupts must be enabled within the AT6nnn.
2. Appropriate PC-AT hardware interrupt must be enabled within the 8259 interrupt controller.
3. The interrupt flag must be set within the 80x86 processor.

When the AT6nnn interrupt arrives at the 80x86 processor, the address of the interrupt service routine must be in the interrupt vector table so that the 80x86 processor knows where to handle the interrupt.

AT6nnn Interrupt Enable

Up to four different kinds of interrupts can be enabled within the AT6nnn:

- Interrupt PC-AT when AT6nnn output buffer has data
- Interrupt PC-AT when AT6nnn input buffer (256 bytes/characters) is empty
- Interrupt PC-AT when AT6nnn hardware interrupt condition occurs (see description of `INTHW` command in ***6000 Series Software Reference***)
- Interrupt PC-AT when AT6nnn status has been updated

AT6nnn interrupts are enabled/disabled by writing to the AT6nnn interrupt enable register (AT6nnn base port address + 4).

Also, DIP switch package S2 on the AT6nnn card must be set to the appropriate hardware interrupt request line (IRQ0-IRQ15). Refer to the table below. IRQ0-IRQ15 are signals that reside on the PC-AT bus. Setting a switch to the ON position connects the AT6nnn interrupt line to a PC-AT interrupt request (IRQ) line. *Make sure only one switch is ON at one time.*

Interrupt Request Line	AT6nnn Interrupt DIP Switch
IRQ3	S2.1
IRQ4	S2.2
IRQ5	S2.3
IRQ7	S2.4
IRQ10	S2.5
IRQ11	S2.6
IRQ12	S2.7
IRQ15	S2.8

Refer to your product's ***Installation Guide*** to locate the DIP switch.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

8259 Interrupt Enable

The PC-AT has two 8259 interrupt controllers that allow up to 15 different hardware devices to interrupt the PC-AT. The first 8259 (at base port address 20H) handles interrupts IRQ0-IRQ7. The second 8259 (at base port address A0H) handles interrupts IRQ8-IRQ15. The table below lists the PC-AT hardware interrupts by precedence (from highest to lowest priority). Programming of the 8259 controller establishes this precedence.

PC-AT hardware interrupts (IRQ0-IRQ15) are enabled/disabled by writing to the 8259 interrupt mask register (8259 base port address + 1).

PC-AT Hardware Interrupt (highest to lowest priority)	Interrupt Function
IRQ0 (highest)	Timer
IRQ1	Keyboard
IRQ2	Reserved
IRQ8	Real-time clock
IRQ9	Reserved
IRQ10	Reserved
IRQ11	Reserved
IRQ12	Reserved
IRQ13	Math co-processor
IRQ14	Hard disk controller
IRQ15	Reserved
IRQ3	COM2 serial
IRQ4	COM1 serial
IRQ5	LPT2 printer
IRQ6	Floppy disk controller
IRQ7 (lowest)	LPT1 printer

80x86 Maskable Interrupt Enable

The 80x86 processor is capable of disabling all maskable hardware interrupts (IRQ0-IRQ15). However, it cannot disable a non-maskable interrupt (NMI), such as a memory parity error interrupt or a divide-by-zero interrupt.

You may want to disable interrupts to protect portions of code from reentrance. For example, if a function is being executed at the time an interrupt occurs and an interrupt service routine (ISR) is called, the ISR may attempt to execute that same function. If this happens, the function is said to be *reentered*. Keep in mind that most INT 21H functions should not be called within an ISR because most DOS functions are non-reentrant.

PC-AT maskable hardware interrupts (IRQ0-IRQ15) are enabled/ disabled by the 80x86 instructions `STI` and `CLI`. `STI` sets the 80x86 interrupt flag, enabling interrupts. `CLI` clears the 80x86 interrupt flag, disabling interrupts.

80x86 Interrupt Vectors

Interrupt vectors tell the 80x86 processor where to go to handle an interrupt. Interrupt vectors are stored in low memory. Because the 80x86 processor works with memory in 64K-byte chunks called *segments*, each vector comprises two words: a segment value and an offset into the segment.

The table below shows the vector address table for PC-AT hardware interrupts. If you decide to let the AT6nnn use IRQ5, for example, then you must replace the vector at address 0034-0037 with the vector of your interrupt service routine (ISR).

Interrupt Number	Vector Address	Interrupt Description
08	0020 - 0023	IRQ0 (Timer)
09	0024 - 0027	IRQ1 (Keyboard)
0A	0028 - 002B	IRQ2 (Reserved)
0B	002C - 002F	IRQ3 (COM2)
0C	0030 - 0033	IRQ4 (COM1)
0D	0034 - 0037	IRQ5 (LPT2)
0E	0038 - 003B	IRQ6 (Floppy disk)
0F	003C - 003F	IRQ7 (LPT1)
70	01C0 - 01C3	IRQ8 (Real-time clock)
71	01C4 - 01C7	IRQ9 (Reserved)
72	01C8 - 01CB	IRQ10 (Reserved)
73	01CC - 01CF	IRQ11 (Reserved)
74	01D0 - 01D3	IRQ12 (Reserved)
75	01D4 - 01D7	IRQ13 (Math co-processor)
76	01D8 - 01DB	IRQ14 (Hard disk)
77	01DC - 01DF	IRQ15 (Reserved)

Fortunately, there are DOS services that permit you to easily modify the interrupt vector table.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

How to Use Interrupts

This section leads you through the following steps for using interrupts:

① Install address of interrupt service routine (ISR) in 80x86 interrupt vector table
② Enable hardware interrupts (IRQ0-IRQ15) within 8259 interrupt controller
③ Enable interrupt sources within AT6nnn
④ Process interrupts in ISR
   a. Identify interrupt
   b. Process interrupt
   c. Clear interrupt
   d. Send end-of-interrupt code to 8259 interrupt controller
⑤ Disable interrupt sources within AT6nnn
⑥ Restore original interrupt vector
⑦ Exit program

① Initialize Interrupt Vector

In this step, you will install the address of your interrupt service routine in the 80x86 interrupt vector table.

Use function 35H of INT 21H to retrieve the interrupt vector that is going to be changed. **Save this vector**—later, you will restore the interrupt vector prior to exiting your program (see below).

```
/* Get and save original interrupt vector */
inregs.h.ah     = 0x35;                 /* Function 35H         */
inregs.h.al     = int_num;              /* Interrupt number     */
intdosx(&inregs, &outregs, &segregs);   /* Call INT 21H         */
oldseg          = segregs.es            /* Save vector segment  */
oldoff          = outregs.x.bx;         /* Save vector offset   */
```

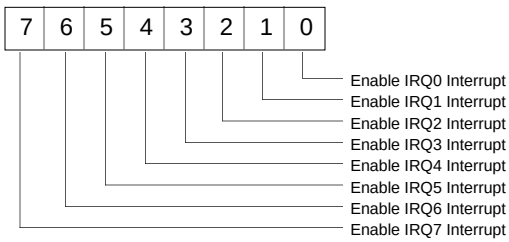
Then use Function 25H of INT 21H to set an interrupt vector to your interrupt service routine. Function 25H automatically disables hardware interrupts when the vector is changed.

```
/* Set interrupt vector to isr */
inregs.h.ah     = 0x25;                 /* Function 25H         */
inregs.h.al     = int_num;              /* Interrupt number     */
segregs.ds      = FP_SEG(isr);          /* Get isr segment      */
inregs.x.dx     = FP_OFF(isr);          /* Get isr offset       */
intdosx(&inregs, &outregs, &segregs);   /* Call INT21H          */
```

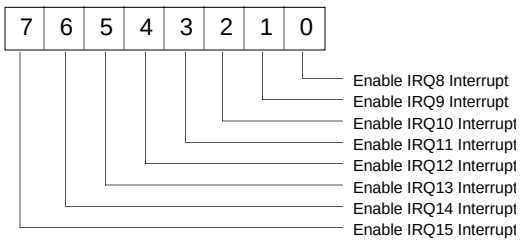
② Enable Interrupts in 8259

This second step requires you to enable a hardware interrupt (IRQ0-IRQ15) within the 8259 interrupt controller. This is accomplished by writing to the 8259 interrupt mask register (8259 base port address + 1).

Interrupt Mask Register (21H) for First 8259



Interrupt Mask Register (A1H) for Second 8259



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Setting a bit to zero (0) will enable a particular hardware interrupt. For example, to enable the IRQ5 interrupt:

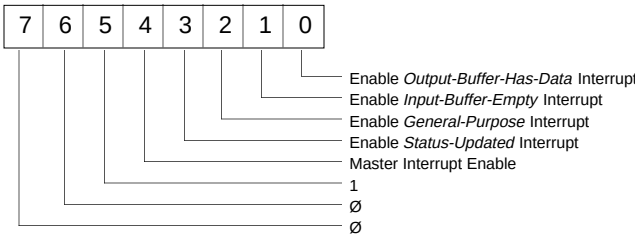
```
#define IRQ5_MASK Ø x2Ø   /* Bit 5 of 8259 IMR */
outp(0x21, inp(0x21) & ~IRQ5_MASK);  /* Enable IRQ5 interrupt */
```

Because interrupts on the second 8259 controller are software redirected to IRQ2 on the first 8259, we must also enable the IRQ2 interrupt for these interrupts. For example, to enable the IRQ11 interrupt:

```
#define IRQ2_MASK  Ø x Ø4   /* Bit 2 on first 8259 IMR  */
#define IRQ11_MASK Ø x 8Ø   /* Bit 3 on second 8259 IMR */
outp(0xA1, inp(0xA1) & ~IRQ11_MASK);  /* Enable IRQ11 interrupt */
outp(0x21, inp(0x21) & ~IRQ2_MASK);   /* Enable IRQ2 interrupt  */
```

③ Enable Interrupts in the AT6nnn

In this third step, you must enable one or more interrupts sources within the AT6nnn. This is accomplished by writing to the AT6nnn interrupt enable register (AT6nnn base port address + 4). Refer to the illustration below.



Setting a bit to 1 will enable a particular AT6nnn interrupt. For example, to enable the general purpose interrupt and status updated interrupt:

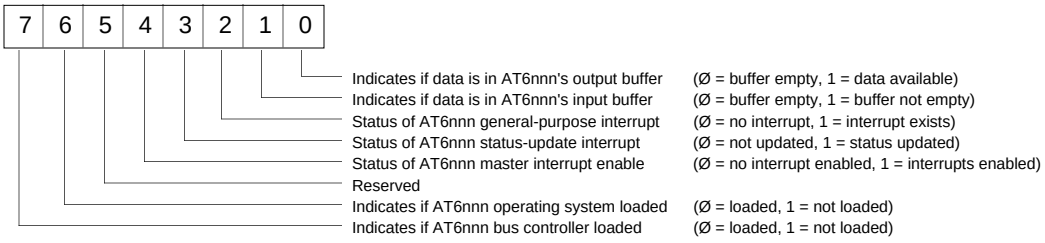
```
#define GP_INTERRUPT    0x04    /* Bit 2 */
#define STATUS_UPDATED  0x08    /* Bit 3 */
#define MASTER          0x10    /* Bit 4 */
int_enable = 0x20;              /* Bit 5 */
int_enable = int_enable | MASTER | GP_INTERRUPT | STATUS_UPDATED;
outp(address+4, int_enable);
```

NOTE

You must keep track of what you write to the AT6nnn interrupt enable register. If you attempt to read this register, it will not return the status of the interrupt enables. Rather, it will return the status of each AT6nnn interrupt and more (see ④ *Process Interrupts in ISR* below).

④ Process Interrupts in ISR

After enabling the interrupts in the AT6nnn, you must write an interrupt service routine (ISR) to identify an interrupt and process it. Interrupts can be identified by reading the AT6nnn status register (AT6nnn base address + 4).



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Bits 0, 1, 2 and 3 of the AT6nnn status register can be used to identify the interrupt. For example, if the AT6nnn general-purpose interrupt is enabled, then check bit 2 of the AT6nnn status register to see if a general-purpose interrupt occurred.

```
/* test for AT6nnn general purpose interrupt */
if(int_enable & GP_INTERRUPT) && (inp(address+4) & GP_INTERRUPT))
{
process_gpint();  /* process the interrupt */
outp(address+4, 0x44);   /* clear the interrupt */
}
```

Important Considerations

Keep in mind that when processing interrupts, you should minimize the amount of time spent in the interrupt service routine. If you do not, your background processing will suffer.

The interrupt must be cleared prior to exiting the ISR. If you do not clear the interrupt, the PC-AT will be continually interrupted by the same interrupt.

The AT6nnn output-buffer-has-data interrupt is cleared simply by fetching data from the AT6nnn output data buffer. Likewise, the AT6nnn input-data-buffer-empty interrupt is cleared by sending data to the AT6nnn input data buffer and a "`data ready`" command (0x42) to the AT6nnn status register.

The AT6nnn general-purpose interrupt is cleared by sending a "`clear general purpose interrupt`" command (0x44) to the AT6nnn status register.

The AT6nnn status-update interrupt is cleared by requesting another status update via the "`update status request`" command (0x48).

Before exiting the ISR, you must send an "`end-of-interrupt`" command (0x20) to the 8259 command register. If you are using an interrupt on the second 8259, you must send 0x20 to both 8259s in the PC-AT (see below).

```
/* send EOI to 8259 */ outp(0x20, 0x20);
```

⑤ Disable Interrupts in the AT6nnn

You must next disable all interrupt sources from the AT6nnn. This is accomplished by writing to the AT6nnn interrupt enable register (AT6nnn base port address + 4). Setting the Master Interrupt Enable bit to Ø will disable all AT6nnn interrupts (as follows).

```
#define MASTER  0x10   /* Master Interrupt Enable mask */
int_enable = int_enable & ~MASTER;
outp(address+4, int_enable);
```

⑥ Restore Original Interrupt Vector

Before exiting your program, use function 25H of INT 21H to restore the original interrupt vector that was previously saved at the beginning of your program.

```
/* Restore original interrupt vector */
inregs.h.ah = 0x25;                    /* Function 25H      */
inregs.h.al = int_num;                 /* Interrupt number  */
segregs.ds  = oldseg;                  /* Get segment       */
inregs.x.dx = oldoff;                  /* Get offset        */
intdosx(&inregs, &outregs, &segregs);  /* Call INT21H       */
```

⑦ Exit Program

If you have completed steps 1 through 6 above, you may now exit the program.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Interrupt-Driven Terminal Emulator

`MC6ØTRMI.EXE` is an interrupt-driven terminal emulator for the AT6nnn that is provided (along with source code) on the AT6nnn DOS Support Disk. This software, written in Microsoft C 6.0, shows how to exploit three of the four available AT6nnn interrupts.

- AT6nnn output-buffer-has-data interrupt
- AT6nnn input-buffer-is-empty interrupt
- AT6nnn general purpose interrupt (see `INTHW` command)

The file `MC6ØTRMI.C` contains the background polling loop, and the file `MC6ØLIBI.C` contains the AT6nnn interrupt driver. *The AT6nnn status-update interrupt is not used in* `MC6ØTRMI`.

The background polling loop (`Emulate()`) continually checks for keyboard input, AT6nnn response data, and the general-purpose interrupt flag. Keyboard data is sent to the AT6nnn upon receipt of a carriage return. AT6nnn response data and AT6nnn general-purpose interrupts are displayed on the screen.

The AT6nnn interrupt driver consists of the ISR (`AT6nnn_isr()`) and additional ring buffer and interrupt vector management functions.

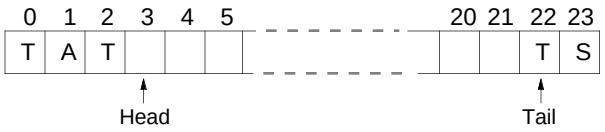
Ring Buffers

To facilitate interrupt-driven communications, *ring buffers* (or circular buffers) are used as the interface between background and foreground processing. A ring buffer is needed because data is coming into the buffer at a different rate than it is going out. This is due to the background polling rate being different than the rate at which the PC-AT is being interrupted.

A ring buffer is nothing more than a data structure with three components: a *buffer*, a *head pointer*, and a *tail pointer*. The head pointer points to the next buffer position that you can put data into. The tail pointer points to the oldest data item in the buffer.

When the head and tail pointers are equal, the ring buffer is empty. The ring buffer is full if the addition of another data item would make the head and tail pointers equal. When either the head or tail pointer goes beyond the end of the buffer, it wraps to the beginning of the buffer (thus the name, *ring* buffer).

The illustration below shows a 24-byte ring buffer containing the command `TSTAT`. Notice that the command starts in position 22, wraps around to 0 and continues until it is complete.



The AT6nnn interrupt driver in `MC60LIBI.C` maintains two ring buffers: an input ring buffer and an output ring buffer.

Input Ring Buffer:

When the AT6nnn has response data, an interrupt is generated and the interrupt service routine stuffs the response data into an input ring buffer. If the input ring buffer becomes full, the interrupt service routine will turn off the AT6nnn output-buffer-has-data interrupt.

The background polling loop fetches the AT6nnn response data from the input ring buffer, displays the data on the screen, and re-enables the AT6nnn output-buffer-has-data interrupt if it was previously disabled.

Output Ring Buffer:

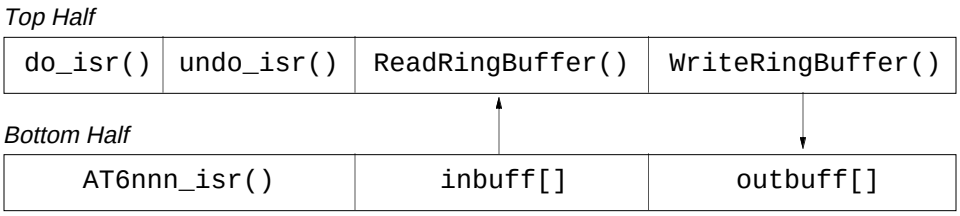
When the background polling loop has command data to send to the AT6nnn, it stuffs the command data into an output ring buffer and enables the AT6nnn input-buffer-is-empty interrupt.

Once the AT6nnn input buffer is empty, an interrupt is generated and the interrupt service routine fetches the command data from the output ring buffer, sends the data to the AT6nnn, and disables the AT6nnn input-buffer-is-empty interrupt.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

AT6nnn Interrupt Driver

The AT6nnn interrupt driver can be thought of as having a *top half* and a *bottom half* (see illustration below).



The *top half* consists of those functions that can be called from your program (see table below).

Function	Description
`do_isr()`	Initializes interrupt vector, and enables PC-AT and AT6nnn interrupts
`undo_isr()`	Disables AT6nnn interrupts and restores original interrupt vector
`ReadRingBuffer()`	Used to read AT6nnn response data from input ring buffer
`WriteRingBuffer()`	Used to write AT6nnn command data into output ring buffer

The *bottom half* is called on an interrupt issued by the AT6nnn (see below).

Function	Description
`AT6nnn_isr()`	Interrupt service routine
`inbuff[]`	Input ring buffer
`outbuff[]`	Output ring buffer

You may have noticed that a ring buffer was not set up to handle the AT6nnn general-purpose interrupt. Instead, a flag is set in the interrupt service routine upon receipt of the general purpose interrupt. Seeing this flag set, the background polling loop displays the interrupt status bits and clears the flag. You can just as easily set up another ring buffer to collect the interrupt status bits.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Controlling Multiple Serial Ports

Every stand-alone 6000 Series product has two serial ports. On existing 6000 products, the RS-232 connector (or Rx, Tx, and GND terminals on an AUX connector) is referenced as the "COM1" serial port, and the RP240 connector is referenced as the "COM2" serial port. Newer products have connectors labeled "COM1" (factory default function is RS-232) and "COM2" (factory default function is RP240). New features were added in software revision 4.0 to allow greater flexibility for the two serial ports:

- Beginning of transmission characters may now be specified (with the BOT command) for all responses from the 6000 product.
- The XONOFF command was created to enable or disable XON/XOFF ASCII handshaking. (XONOFF1 enables XON/XOFF, XONOFFØ disables XON/XOFF) Defaults: XONOFF1 for the COM1 port, XONOFFØ for the COM2 port.

  Controllers on a multi-drop do not support XON/XOFF; to ensure that XON/XOFF is disabled for COM2, send the PORT2 command followed by the XONOFFØ command.
- Several commands were added to control communication on both serial communication ports on all stand-alone products (see *Configuring the COM Ports* below for details).
- Support for RS-485 4-wire multi-drop communication, (see *RS-485 Multi-Drop*, page 75, for details).

Configuring the COM Port

To control the applicable port for setting up serial communication and transmitting ASCII text strings, use the PORT command. PORT1 selects COM1 and PORT2 selects COM2.

- Serial communication setup commands (see list below) affect the COM port selected with the last PORT command. For example, to configure the COM2 port for 6000 language commands only (e.g., to communicate to the 6000 product over an RS-485 interface), execute the PORT2 command, then execute the DRPCHKØ command.

  DRPCHK.....RP240 Check
  E.....Enable Serial Communication
  ECHO.....Enable Communication Echo
  BOT.....Beginning of Transmission Characters
  EOT.....End of Transmission Characters
  EOL.....End of Line Terminating Characters
  ERRBAD.....Error Prompt
  ERRDEF.....Program Definition Prompt
  ERRLVL.....Error Detection Level
  ERRORK.....Good Prompt
  XONOFF.....Enable or disable XON/XOFF

- The PORT command also selects the COM port through which the WRITE and READ commands transmit ASCII text strings. If an RP240 is connected, the DWRITE command (and 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.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Setup for 6000 Language or RP240

To configure the COM ports for use with 6000 language commands or an RP240, use the DRPCHK command. The DRPCHK command affects the COM port selected with the last PORT command. The default for COM1 is DRPCHKØ; the default for COM2 is DRPCHK3. The DRPCHK setting is automatically saved in non-volatile memory. NOTE: Only one COM port may be set to DRPCHK2 or DRPCHK3 at any given time.

DRPCHKØ .....Use the COM port for 6000 language commands only. 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 .....Check for the presence of an RP240 at power-up/reset. If an RP240 is present, initialize the RP240. If an RP240 is not present, use the port only for 6000 language commands. NOTE: RP240 commands will be sent at power-up and reset.

DRPCHK2 .....Check for the presence of an RP240 every 5-6 seconds. If an RP240 is plugged in, initialize the RP240.

DRPCHK3 .....Check for the presence of an RP240 at power-up/reset. If an RP240 is present, the initialize the RP240. If an RP240 is not present, use the COM port for DWRITE commands only, and 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).

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.

Selecting a Destination Port for Transmitting from the Controller

To define the port (COM port) through which the 6000 product sends its responses, you have 3 options:

- Do nothing different. The response will be sent to the COM port through which the request was made. If the command is in a stored program, the report will be sent to the COM port selected by the most recent PORT command.
- Prefix the command with [. This causes the response to be sent to both COM ports. (e.g., the [TFS command response will be sent through both COM ports)
- Prefix the command with]. This causes the response to be sent to the alternative COM port. For example, if a report back (e.g.,]TAS) is requested from COM1, the response is sent through COM2. If the command is in a stored program, the report will be sent out the alternate port from the one selected by the most recent PORT command.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

RS-232C Daisy-Chaining

Up to ninety-nine stand-alone 6000 Series products may be daisy-chained. There are two methods of daisy-chaining: one uses a computer or terminal as the controller in the chain; the other uses one 6000 product as the master controller. Refer to you product's ***Installation Guide*** for daisy-chain connections.

Follow these steps to implement daisy-chaining:

Step 1 To enable and disable communications on a particular controller unit in the chain, you must establish a unique device address using the unit's address DIP switches or the Daisy-chain Address (ADDR) command.

DIP switches: Instructions for accessing and changing these DIP switch settings are provided in your controller's ***Installation Guide***. Device addresses set with the DIP switches range from 0 to 7.

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

Sending ADDRi to the first unit in the daisy 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).

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

Note that a controller with the default device address of zero (0) will send an initial power-up start message similar to the following:

```
*PARKER 6nnn MOTION CONTROLLER
*NO REMOTE PANEL
```

Step 2 Connect the daisy-chain with a terminal as the master (see diagram in the product's ***Installation Guide***).

It is necessary to have the error level set to 1 for all units on the daisy-chain (ERRLVL1). When the error level is not set to 1, the controller sends ERROK or ERRBAD prompts after each command, which makes daisy-chaining impossible. Send the ERRLVL1 command to each unit in the chain. (NOTE: To send a the ERRLVL1 command to one specific unit on the chain, prefix the command with the appropriate unit's device address and an underline.)

Commands:

```
1_ERRLVL1      ; Set error level to 1 for unit #1
2_ERRLVL1      ; Set error level to 1 for unit #2
3_ERRLVL1      ; Set error level to 1 for unit #3
```

After this has been accomplished, a carriage return sent from the terminal will not cause any controller to send a prompt. Verify this. Instructions below (step 3) show how to set the error level to 1 automatically on power-up by using the controller's power-up start program (highly recommended).

After the error level for all units has been set to ERRLVL1, send a 6000 series command to all units on the daisy-chain by entering that command from the master terminal.

Commands:

```
OUT1111        ; Turn on outputs #1 - #4 on all units
A50,50         ; Set acceleration to 50 for all axes (all units, both axes)
```

To send a 6000 series command to one particular unit on the chain, prefix the command with the appropriate unit's device address and an underline:

Commands:

```
2_OUT0         ; Turn off output #1 on unit #2
4_OUT0         ; Turn off output #1 on unit #4
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

To receive data from a particular controller on the chain, you **must** prefix the command with the appropriate unit's device address and an underline:

Commands:
```
1_A             ; Request acceleration information from unit #1
*A50,50         ; Response from unit #1
```

Use the E command to enable/disable RS-232C communications for an individual unit. If all 6000 controller units on the daisy chain are enabled, commands without a device address identifier will be executed by all units. Because of the daisy-chain's serial nature, the commands will be executed approximately 1 ms per character later on each successive unit in the chain (assuming 9600 baud).

Units with the RS-232C disabled (EØ) will not respond to any commands, except E1; however, characters are still echoed to the next device in the daisy chain.

Commands:
```
3_E0            ; Disable RS-232C on unit #3
VAR1=1          ; Set variable #1 to 1 on all other units
3_E1            ; Enable RS-232C on unit #3
3_VAR1=5        ; Set variable #1 to 5 on unit #3
```

Verify communication to all units by using the techniques described above.

Step 3 Now that communication is established, programming of the units can begin (alternatively, units can be programmed individually by connecting the master terminal to one unit at a time). To allow daisy-chaining between multiple controllers, the ERRLVL1 command must be used to prevent units from sending error messages and command prompts. In every daisy-chained unit, the ERRLVL1 command should be placed in the program that is defined as the STARTP program:

Program:
```
DEF chain       ; Begin definition of program chain
ERRLVL1         ; Set error level to 1
GOTO main       ; Go to program main
END             ; End definition of program chain
STARTP chain    ; Designates program chain as the power-up program
```

To define program main for unit #0:

Program:
```
0_DEF main      ; Begin definition of program main on unit #0
0_GO            ; Start motion
0_END           ; End definition of program main on unit #0
```

Step 4 After all programming is completed, program execution may be controlled by either a master terminal, or by a 6000 Series controller used as a master.

Daisy-Chaining from a Computer or Terminal

Controlling the daisy-chain from a master computer or terminal follows the examples above:

Commands:
```
0_RUN main      ; Run program main on unit #0
1_RUN main      ; Run program main on unit #1
2_GO1           ; Start motion on unit #2 axis #1
3_2A            ; Get A command response from unit #3 axis #2
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Daisy-Chaining from a Master 6000 Controller

Controlling the daisy-chain from a master 6000 controller (the first unit on the daisy-chain) requires the programs stored in the master controller to control program and command execution on the slave controllers. The example below demonstrates the use of the WRITE command to send commands to other units on the daisy chain.

> **NOTE**
> The last unit on the daisy-chain must have RS-232C echo disabled (ECHOØ command).

Master controller's main program:

Program:

```
DEF main                ; Program main
L                       ; Indefinite loop
 WHILE (IN.1 = b0)      ; Wait for input #1 to go active
 NWHILE
 GOL                    ; Initiate linear interpolated move
 WHILE (IN.1 = b1)      ; Wait for input #1 to go inactive
 NWHILE
 WRITE"2_D2000,4000"    ; Send message "2_D2000,4000" down the daisy chain
 WRITE"2_ACK"           ; Send message "2_ACK" down the daisy chain
LN                      ; End of loop
END                     ; End of program main
```

Controller unit #2 ack program:

Program:

```
DEF ack                 ; Program ack
GO11                    ; Start motion on both axes
END                     ; End of program ack
```

Daisy-Chaining and RP240s

RP240s cannot be placed in the controller daisy chain; RP240s can only be connected to the designated RP240 port on a controller. It is possible to use only one RP240 with a controller daisy-chain to input data for multiple units on the chain. The example below (for the controller master with an RP240 connected) reads data from the RP240 into variables #1 (*data1*) & #2 (*data2*), then sends the messages 3_Ddata1,data2<CR> and 3_GO<CR>.

Sample portion of code:

```
L                       ; Indefinite loop
 VAR1=DREAD             ; Read RP240 data into variable #1
 VAR2=DREAD             ; Read RP240 data into variable #2
 EOT0,0,0,0             ; Turn off <CR>
 WRITE"3_D"             ; Send message "3_D" down the daisy chain
 WRVAR1                 ; Send variable #1 data down the daisy chain
 WRITE","               ; Send message "," down the daisy chain
 EOT13,0,0,0            ; Turn on <CR>
 WRVAR2                 ; Send variable #2 data down the daisy chain
 WRITE"3_GO"            ; Send message "3_GO" down the daisy chain
LN                      ; End of loop
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

RS-485 Multi-Drop (RS-485 Compatible Products Only)

Up to 99 stand-alone 6000 Series products may be multi-dropped. Refer to your product's ***Installation Guide*** for multi-drop connections.

To establish device addresses, using the ADDR command (the 6104 also allows you to set the device address with DIP switches, providing up to 31 unique addresses):

The ADDR command allows you to establish up to 99 unique addresses. 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.

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

NOTE

Controllers on a multi-drop do not support XON/XOFF. To ensure that XON/XOFF is disabled for COM2, send the PORT2 command followed by the XONOFFØ command.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

CHAPTER THREE

3 Basic Operation Setup

IN THIS CHAPTER

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

- Before You Begin (setup programs, Motion Architect, resetting, etc.) 78
- Participating Axes 79
- Memory Allocation 80
- Drive Setup 80
- Axis Scaling 83
- Positioning Modes 87
- End-of-Travel Limits 90
- Homing 91
- Closed Loop Stepper Setup (steppers only) 95
- Servo Setup (servos only) 98
- Target Zone Mode 105
- Programmable Inputs and Outputs (incl. triggers and auxiliary outputs) 106
- Variable Arrays (*teaching variable data*) 120

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

> **WARNING**
>
> The 6000 Product is used to control your system's electrical and mechanical components. Therefore, you should test your system for safety under all potential conditions. Failure to do so can result in damage to equipment and/or serious injury to personnel.

Setup Parameters Discussed in this Chapter

Other status commands are described on page 232.

Below is a list of the setup parameters discussed in this chapter. You can check the status of each parameter setting by entering the respective setup command without any command fields (e.g., typing INFNC <*cr*> displays the current function and state of each programmable input). Some setup parameters are also reported with the TSTAT and TASF status commands (see page 232).

Setup Parameter	Command	See Pg.
Participating Axes *	INDAX	79
Memory (status with TDIR & TMEM)	MEMORY	12 & 80
Drive Setup		80
Drive Fault Level	DRFLVL	
Drive Resolution *	DRES	
Step Pulse Width	PULSE	
Start/Stop Velocity	SSV	
Drive Disable on Kill	KDRIVE	
ZETA6104 Drive Setup:		
Motor inductance	DMTIND	
Motor static torque	DMTSTT	
Activate damping	DACTDP	
Anti-resonance	DAREN	
Electronic viscosity	DELVIS	
Automatic current reduction	DAUTOS	
Axis Scaling		83
Enable scaling factor *	SCALE	
Acceleration scaling factor *	SCLA	
Distance scaling factor *	SCLD	
Velocity scaling factor *	SCLV	
Positioning Mode		87
Continuous or preset **	MC	
Preset: absolute or incremental **	MA	
End-of-travel limits		90
Hardware – enabled *	LH	
Hardware – deceleration	LHAD	
Hardware – s-curve decel (servos)	LHADA	
Hardware – active level of input	LHLVL	
Software – enabled *	LS	
Software – deceleration	LSAD	
Software – s-curve decel (servos)	LSADA	
Software – negative direction limit	LSNEG	
Software – positive direction limit	LSPOS	
Homing		91
Acceleration	HOMA	
S-curve acceleration (servos only)	HOMAA	
Deceleration	HOMAD	
S-curve deceleration (servos only)	HOMADA	
Backup to home	HOMBAC	
Final approach direction	HOMDF	
Stopping edge of switch	HOMEDG	
Home switch active level	HOMLVL	
Velocity	HOMV	
Velocity of final approach	HOMVF	
Home to Z channel input	HOMZ	

Setup Parameter	Command	See Pg.
Closed-loop Stepper Setup (steppers only)		95
Motor (drive) resolution *	DRES	
Encoder resolution *	ERES	
Encoder/motor step mode select **	ENC	
Position maintenance **	EPM	
Stall detection **	ESTALL	
Kill on stall detected	ESK	
Stall deadband	ESDB	
Use encoder as counter	CNTE	
Encoder polarity	ENCPOL	
Commanded direction polarity	CMDDIR	
Servo Setup (servos only)		98
Tuning parameters	(see page 99)	
Feedback source selection	SFB	
Update rates	SSFR	
Maximum position error	SMPER	
DAC output limit, maximum	DACLIM	
DAC output limit, minimum	DACMIN	
Dither, amplitude	SDTAMP	
Dither, frequency ratio	SDTFR	
Feedback polarity, encoder	ENCPOL	
Feedback polarity, ANI input	ANIPOL	
Feedback polarity, LDT	LDTPOL	
Commanded direction polarity	CMDDIR	
Servo control signal offset	SOFFS	
Servo control signal offset, negative	SOFFSN	
Setpoint window, distance	SSWD	
Setpoint window, gain set	SSWG	
Target Zone (end-of-move settling criteria)		105
Target zone mode enable	STRGTE	
Target distance zone	STRGTD	
Target velocity zone	STRGTV	
Target settling timeout period	STRGTT	
Programmable Input Functions		106
Enable input functions *	INFEN	
Define input functions *	INFNC	
Input active level	INLVL	
Input debounce	INDEB	
Trigger input functions	TRGFN	
Programmable Output Functions		106
Enable output functions *	OUTFEN	
Define output functions *	OUTFNC	
Output active level	OUTLVL	
Variable Arrays (*teaching* variable data)		120
Initialize numeric variable for data	VAR	
Define data program and program size	DATSIZ	
Set data pointer & establish increment	DATPTR	
Reset data pointer to specific location	DATRST	

* You can also check the status with the TSTAT status command.
** You can also check the status with the TASF status command.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

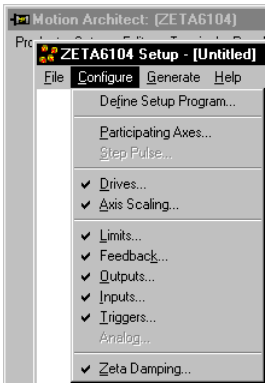
Using a Setup Program

The features described in this chapter are configured with certain 6000 Series commands, commonly referred to as "setup commands." We recommend placing these commands (except MEMORY) into a special "setup program" that is executed to prepare the 6000 Series product for subsequent controller operations. Further details about setup programming is provided in the *Creating and Executing a Setup Program* section, page 14.

Motion Architect

Use Motion Architect's Setup module to help you create the basic configuration program. By simply responding to a series of dialog boxes, a program is created with a specific name (as if you created it in the usual process with the DEF and END commands, as demonstrated on page 9).

You can make any necessary modifications to the setup program file in the Editor Module, and then download the file to the 6000 Series controller from the Terminal Module, or test it with the Panel Module.



Resetting the Controller

Bus-based controllers: The RESET command returns all previously entered command parameters to their original factory default values. **CAUTION**: all programs and subroutines will be deleted.

Stand-alone (serial-based) controllers: The RESET command acts the same as cycling power. All programs and variables, and some command settings (see page 33), will be retained; all other previously entered command parameters not saved in a program or a variable will be returned to their default values. If you are using an RP240, the RESET function is available if you use the default menu system (see page 137).

Participating Axes

If you are not going to use all the axes available to you by your 6000 Series product, use the INDAX command to remove from service the unnecessary axes. For example, if you have a 4-axis controller and will use only 3 axes, issue the INDAX3 command and the controller will function as if axis #4 was deleted from its design.

No report-backs or command parameters are accepted for axes excluded as a result of the INDAX command. For instance, if you specify INDAX3 (use axes 1-3 only), the A command would show a response of *A10.0000,10.0000,10.0000, and if you tried to enter a command value for the fourth axes (e.g., 4A30), you would receive the error response "*INCORRECT AXIS".

Servo Users

Changing the INDAX setting also changes the update rates (servo, motion trajectory, and system). For details, refer to the SSFR command description in the ***6000 Series Software Reference***.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Memory Allocation

For details about memory allocation, refer to the *Storing Programs* on page 12.

CAUTION

Issuing a new MEMORY command (e.g., MEMORY30000,10000) will erase all existing programs and compiled contouring path segments residing in the 6000 product's memory. To determine the status of memory allocation, use the TMEM command.

If you are using a stand-alone serial product, do not place the MEMORY command in the program assigned as the startup (STARTP) program. Doing so would erase all programs and segments upon cycling power or issuing the RESET command.

Drive Setup

Drive Fault Level

This setup parameter is not required for packaged drive/controller products (e.g., 610n, 6201).

The drive fault level (DRFLVL) should be set to "active high" or "active low" for each axis. The drive fault input schematic is shown in your 6000 Series product installation guide. Use the table below as a guide (the drive fault level for packaged controller/drive products is factory set). NOTE: The drive fault input is not available on the OEM-AT6400 product.

Compumotor Product	Drive Fault Level
BLH, L, LE, PDS, PK130	Active Low (DRFLVLØ)
Apex Series, CD (in CN rack), Dynaserv , LN, OEM Series, S, SD (in SC rack), TQ, Z, Zeta	Active High (DRFLVL1)

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

NOTE

Once the drive fault level has been configured, you must enable the drive fault input with the INFEN1 command before the input is usable.

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

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

When a drive fault occurs, motion will be stopped on all axes and program execution will be terminated. The way in which motion is stopped varies between servo and stepper products:

Servos: Motion is stopped at the rate set with the LHAD command (default is 100 units/sec^2).

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

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Drive Resolution *(steppers only)*

This setup parameter is not required for packaged drive/controller products (e.g., 610n, 615n, 6201).

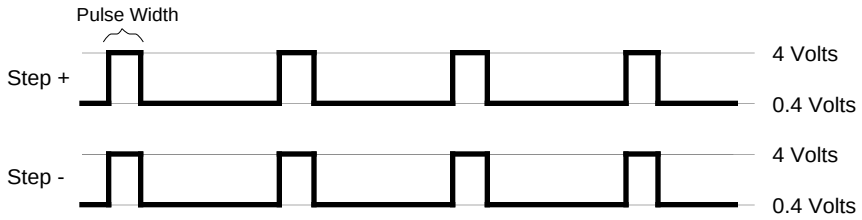
The drive resolution controls the number of steps the stepper controller considers as a full revolution for the motor drive. The controller's resolution is set with the DRES command (default is 25,000 steps/rev). Refer to the user documentation that accompanied your drive if you need to change its resolution.

IMPORTANT NOTES

- If the controller's resolution (set with the DRES command) does not match the drive's resolution, the motor will not move according to the programmed distance and velocity.
- Contouring: All axes involved in contouring (identified with PAXES command) must have the same DRES setting.

Step Pulse *(steppers only)*

The step output pulse width can be varied using the PULSE command. The pulse width can be 0.3 µs to 20 µs (default is 0.3 µs). The pulse width is the amount of time the step output signal is active (see illustration below). The step output pulse width should be configured to meet the minimum step input pulse width requirement of the motor drive you are using.



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

When the pulse width is changed from the default value of 0.3 µs, the maximum velocity 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
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. (This pertains only to bus-based stepper controllers.)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Start/Stop Velocity *(steppers only)*

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 zero 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.

Disable Drive On Kill *(servos only)*

Normally, when you issue a Kill command (`K`, `!K`, or `<ctrl>K`) or activate a general-purpose input configured as a kill input (see `INFNCi-C` command), motion is stopped at the hard limit (`LHAD`/`LHADA`) deceleration setting and the drives are left in the enabled state (`DRIVE1111`).

However, your application may require you to *disable* (*shut down* or *de-energize*) the drives in a Kill situation to, for example, prevent damage to the motors or other system mechanical components. If so, set the controller to the *Disable Drive on Kill* mode with the `KDRIVE1` command. In this mode, a kill command or kill input will shut down the drives immediately, letting the motors *free wheel* (without control from the drives) to a stop. When the drives are disabled (`DRIVEØ`), the **SHTNC** relay output is connected to **COM** and the **SHTNO** relay output is disconnected from **COM**. To re-enable the drives, issue the `DRIVE1` command.

ZETA610n Internal Drive Setup *(610n only)*

Several drive setup parameters are unique to the 610n package drive/controllers. The table below lists the parameters and the relevant setup commands. Refer to the ***ZETA6104 Installation Guide*** for details.

Setup Feature	Description	Command	Default Setting
Current Standby Mode (automatically reduces motor current at rest)	The automatic standby current feature that reduces motor current by 50% if no step pulses have been commanded for a period of 1 second or more. (**WARNING**: torque is also reduced.) Full current is restored upon the first step pulse.	`DAUTOS`	`DAUTOSØ` (current standby mode disabled)
Motor Inductance	Motor inductance setting required for Active Damping feature.	`DMTIND`	`DMTIND1` (range of ≤ 20.08 mH, for 57-51S, 57-83S & 57-102S ZETA motors)
Motor Static Torque	Motor static torque setting required for Active Damping feature.	`DMTSTT`	`DMTSTT1` (range of 0.26-0.72 N-m (36-100oz-in) for 57-51 & 57-83 ZETA motors)
Current Waveform	Sets the waveform of the current from the drive to the motor.	`DWAVEF`	`DWAVEF1` (selects the -4% 3rd harmonic waveform)
Anti-Resonance	General-purpose damping circuit; provides aggressive and effective damping at speeds above 3 revs/second. Disabled is Active Damping is enabled.	`DAREN`	`DAREN1` (enabled)
Active Damping	Extremely power damping circuit for speeds above 3 revs/second. Requires configuration for specific motor size and load (see Installation Guide).	`DACTDP`	`DACTDPØ` (disabled)
Electronic Viscosity	Provides passive damping at speeds below 3 revs/second. Requires configuration for specific application dynamics.	`DELVIS`	`DELVISØ` (disabled)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Axis Scaling

FOLLOWING:
Use the SCLMAS command to establish a distance scale factor for the master axis. Refer to page 196 for details.

The scaling commands allow you to scale acceleration, deceleration, velocity, and position to values that are appropriate for your application. The SCALE, SCLA, SCLV, SCLD, PSCLA, PSCLV, and PSCLD commands are used to implement the scaling features.

Scaling is disabled (SCALEØ) as the factory default condition (exception: enabled for the 6270):

- 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:

	Units of Measure (per feedback source)			
Motion Attribute	**Encoder**	**Resolver**	**ANI**	**LDT**
Accel/Decel	Revs/sec^2 *	Revs/sec^2 *	volts/sec^2	Inches/sec^2 **
Velocity	Revs/sec *	Revs/sec *	volts/sec	Inches/sec **
Distance	Counts (*steps*) ***	Counts (*steps*) ***	Counts (*steps*) ***	Counts (*steps*) ***

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

When Should I Define Scaling Parameters?

To maximize the efficiency of the 6000 product's microprocessor, the scaling multiplications are performed when the program is defined or downloaded. Therefore, you must enable scaling (SCALE1) and define the scaling factors (SCLD, SCLA, SCLV, PSCLA, PSCLV, PSCLD) prior to defining (DEF), uploading (TPROG), or running (RUN) the program.

Users of serial products should put the scaling factors into the startup (STARTP) program. Users of bus-based products should put the scaling factors into a setup program that must be run prior to defining or downloading any other programs. Regardless of the product type, scaling factors could be defined via a terminal emulator just before defining or downloading the program.

Servo Products

Scaling can be used with all feedback sources: encoders, internal resolvers (615n only), ANI inputs (-ANI option only), and LDTs (6270 only). When the scaling commands (SCLA, SCLD, etc.) are executed, they are specific only to the current feedback source selected with the last SFB command.

If your application requires switching between feedback sources for the same axis, then for each feedback source, you must select the feedback source with the appropriate SFB command and issue the scaling factors specific to operating with that feedback source.

For example, if you have two axes and will be switching between encoder and ANI feedback, you should include code similar to the following in your setup program:

```
SFB1,1          ; Select encoder feedback (subsequent scaling
                ; parameters are specific to encoder feedback)
SCLA4000,4000   ; Program accel/decel in revs/sec^2
SCLV4000,4000   ; Program velocity in revs/sec
SCLD4000,4000   ; Program distances in revs
SFB2,2          ; Select ANI feedback (subsequent scaling
                ; parameters are specific to ANI feedback)
SCLA819,819     ; Program accel/decel in volts/sec^2
SCLV819,819     ; Program velocity in volts/sec
SCLD819,819     ; Program distances in volts
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Acceleration & Deceleration Scaling (SCLA and PSCLA)

Steppers: If scaling is enabled (SCALE1), all accel/decel values entered are internally multiplied by the acceleration scaling factor to convert user units/sec^2 to motor steps/sec^2. The scaled values are always in reference to motor steps, not encoder steps, regardless of the ENC command setting.

Servos: If scaling is enabled (SCALE1), all accel/decel values entered are internally multiplied by the acceleration scaling factor to convert user units/sec^2 to encoder, resolver, LDT, or ANI counts/sec^2. This includes all S-curve accel/decel values (servos only).

All accel/decel commands for non-interpolated motion (e.g., A, AD, HOMA, HOMAD, JOGA, etc.) are multiplied by the SCLA command value. All accel/decel commands for linear and circular interpolated motion (e.g., PA, PAD, etc.) are multiplied by the PSCLA command value.

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

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

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

Product	**Min. Accel or Decel (resolution)**	**Max. Accel or Decel**
Steppers	$\frac{0.001 \times DRES}{SCLA}$	$\frac{999.9999 \times DRES}{SCLA}$
Servos	Encoder or resolver feedback: $\frac{0.001 \times ERES}{SCLA}$ LDT feedback: $\frac{0.001 \times LDTRES}{SCLA}$ ANI feedback: $\frac{0.819}{SCLA}$	Encoder or resolver feedback: $\frac{999.9999 \times ERES}{SCLA}$ LDT feedback: $\frac{999.9999 \times LDTRES}{SCLA}$ ANI feedback: $\frac{818999.9181}{SCLA}$

Velocity Scaling (SCLV and PSCLV)

Steppers: If scaling is enabled (SCALE1), all velocity values entered are internally multiplied by the velocity scaling factor to convert user units/sec to motor steps/sec. The scaled values are always in reference to motor steps, not encoder steps, regardless of the ENC command setting.

Servos: If scaling is enabled (SCALE1), all velocity values entered are internally multiplied by the velocity scaling factor to convert user units/sec to encoder, resolver, LDT, or ANI counts/sec.

All velocity commands for non-interpolated motion (e.g., V, HOMV, HOMVF, JOGVH, JOGVL, etc.) are multiplied by the SCLV command value. The velocity command for linear and circular interpolated motion (PV) is multiplied by the PSCLV command value.

As the velocity scaling factor (SCLV/PSCLV) changes, the velocity command's range and its decimal places also change (see table below). A velocity value with greater resolution than allowed will be truncated. For example, if scaling is set to SCLV1Ø, the V9.9999 command would be truncated to V9.9.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

SCLV/PSCLV Value **(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**
$\frac{\left(\frac{8,000,000}{n}\right)}{\text{SCLV}}$ 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: $\frac{1000 \times \text{ERES}}{\text{SCLV}}$ LDT Feedback: $\frac{1000 \times \text{LDTRES}}{\text{SCLV}}$ ANI Feedback: $\frac{1000 \times 819}{\text{SCLV}}$

Distance Scaling (SCLD and PSCLD)

FOLLOWING: The master axis can be scaled with the SCLMAS command, and the slave axis can be scaled with the SCLD command. Refer to page 196 for details.

Steppers: If scaling is enabled (SCALE1), all distance values entered are internally multiplied by the distance scaling factor. Since the SCLD/PSCLD units are in terms of steps/unit, all distances will thus be internally represented in motor steps (if in ENCØ mode) or encoder steps (if in ENC1 mode).

Servos: If scaling is enabled (SCALE1), all distance values entered are internally multiplied by the distance scaling factor. Since the SCLD/PSCLD units are in terms of counts/unit, all distances will thus be internally represented in encoder, resolver, LDT, or ANI counts.

All distance commands for non-interpolated motion (e.g., D, PSET, REG, SMPER) are multiplied by the SCLD command value. Distance commands for linear and circular interpolated motion (e.g., PARCM, PARCOM, PARCOP, PARCP, PLC, PLIN, PRTOL, PWC) are multiplied by the PSCLD command value.

LDT Users—Programming In Inches Or Millimeters

The default SCLD value is 432 (SCLD432,432), which allows LDT users to program in inches. To program in millimeters, use a SCLD value of 17 (SCLD17,17). These factors must be multiplied by the number of recirculations if the LDT uses more than one recirculation.

As the SCLD/PSCLD scaling factor changes, the distance command's range and its decimal places also change (see table below). A distance value with greater resolution than allowed will be truncated. For example, if scaling is set to SCLD4ØØØ, the D1Ø5.2776 command would be truncated to D1Ø5.277.

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

***** 6270: In the table below, shift the decimal place in the “distance range” column one place to the left.

NOTE FRACTIONAL STEP TRUNCATION NOTE

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

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Scaling Examples

Steppers

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

Axis #2 controls a 25,000 step/rev motor/drive system with position feedback from a 1000-line encoder (4,000 steps/rev post quadrature). This axis is using a 10-pitch leadscrew. This axis will be positioned exclusively in the encoder step mode. The user wishes to program motion parameters in inches; therefore, the scale factor calculation is:
4,000 steps/rev x 10 revs/inch = 40,000 steps/inch.

```
ENC01              ; Use motor step positioning for axis #1,
                   ; Use encoder step positioning for axis #2
SCALE1             ; Enable scaling
DRES25000,25000    ; Set drive resolution to 25,000 steps/rev on both axes
ERES,4000          ; Set encoder resolution to 4,000 steps/rev on axis #2
SCLD125000,40000   ; Allow user to enter distance in inches (both axes)
SCLV125000,40000   ; Allow user to enter velocity in inches/sec (both axes)
SCLA125000,40000   ; Allow entering accel/decel in inches/sec/sec (both axes)
```

Servos

Axis #1 controls a 4,000 count/rev servo motor/drive system (using a 1000-line encoder) attached to a 5-pitch leadscrew. The user wants to position in inches; therefore, the scale factor calculation is 4,000 counts/rev x 5 revs/inch = 20,000 counts/inch. Half way through the motion process, axis #1 must switch to ANI feedback for the purpose of positioning to a voltage (scale factor is 819 counts/volt).

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

```
SFB1,1             ; Select encoder feedback for both axes
ERES4000           ; Set encoder res to 4000 steps/rev (post quadrature)
SCALE1             ; Enable scaling
SCLD20000,40000    ; Allow user to enter distance values in inches
SCLV20000,40000    ; Allow user to enter velocity values in inches/sec
SCLA20000,40000    ; Allow user to enter accel/decel values in inches/sec/sec
SFB2               ; Select ANI feedback for axis #1
SCALE1             ; Enable scaling
SCLD819            ; Allow user to enter distance values in volts
SCLV819            ; Allow user to enter velocity values in volts/sec
SCLA819            ; Allow user to enter accel/decel values in volts/sec/sec
SFB1,1             ; Select encoder feedback for both axes (prepare for motion)
```

Hydraulic Control

Axis #1 controls a 4,000 count/rev servo motor/drive system (using a 1000-line encoder). Attached is a 5-pitch leadscrew that the user wants to position in inches (4,000 counts/rev x 5 revs/inch = 20,000 counts/inch scale factor).

Axis #2 controls a servo valve and hydraulic cylinder using position feedback from an LDT with a gradient of 9.0227 µs/inch, and 2 recirculations. The user would like to program in inches (432 LDT counts per inch x 2 recirculations = 864 counts/inch scale factor).

```
SFB1,3             ; Feedback devices: encoder for axis 1, LDT for axis 2
ERES4000           ; Set encoder res to 4000 steps/rev (post quadrature)
LDTGRD,9.0227      ; Set LDT gradient to 9.0227µs/inch
LDTRES,864         ; Set LDT resolution to 864 (accommodates 2 recirculations)
SCALE1             ; Enable scaling
SCLD20000,864      ; Allow user to enter distance values in inches
SCLV20000,864      ; Allow user to enter velocity values in inches/sec
SCLA20000,864      ; Allow user to enter accel/decel values in inches/sec/sec
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Positioning Modes

The 6000 controller can be programmed to position in either the preset (incremental or absolute) mode or the continuous mode. You should select the mode that will be most convenient for your application. For example, a repetitive cut-to-length application requires incremental positioning. X-Y positioning, on the other hand, is better served in the absolute mode. Continuous mode is useful for applications that require constant movement of the load based on internal conditions or inputs, not distance.

Refer also to the Scaling *section above.*

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

Parameter	Units (Unscaled), Steppers	Units (Unscaled), Servos	Unit Scaling Command *
Acceleration	revs/sec^2	encoder/resolver: revs/sec^2 ANI: volts/sec^2 LDT: inches/sec^2	SCLA or PSCLA
Deceleration	revs/sec^2	encoder/resolver: revs/sec^2 ANI: volts/sec^2 LDT: inches/sec^2	SCLA or PSCLA
Velocity	revs/sec	encoder/resolver: revs/sec ANI: volts/sec LDT: inches/sec	SCLV or PSCLV
Distance	steps	counts	SCLD or PSCLD **

* Scaling must first be enabled with the SCALE1 command. PSCLA, PSCLV, and PSCLD are for interpolated moves.
** An axis assigned as a master (for Following) is scaled by the SCLMAS command.

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

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

- Acceleration 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, see *On-The-Fly Motion Profiling* on page 178.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Preset Positioning Mode

A *preset* move is a point-to-point move of a specified distance. You can select preset moves by putting the 6000 controller into preset mode (canceling continuous mode) using the MCØ command. Preset moves allow you to position the motor/load in relation to the previous stopped position (*incremental mode*—enabled with the MAØ command) or in relation to a defined zero reference position (*absolute mode*—enabled with the MA1 command).

Incremental Mode Moves

The incremental mode is the controller's default power-up mode (exception: default for the 6270 hydraulic controller is absolute). When using the Incremental Mode (MAØ), a preset move moves the motor/load the specified distance from its starting position. For example, if you start at position *N*, executing the D6ØØØ command in the MAØ mode will move the motor/load 6,000 units from the *N* position. Executing the D6ØØØ command again will move the motor/load an additional 6,000 units, ending the move 12,000 units from position *N*.

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

Example

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

Absolute Mode Moves

A preset move in the Absolute Mode (MA1) moves the motor/load the distance that you specify from the *absolute zero position*. This is the default positioning mode for 6270 hydraulic controller only.

Establishing a *Zero Position*

One way to establish the zero position is to issue the PSET command when the load is at the location you would like to reference as absolute position zero (e.g., PSETØ,Ø defines the current position as absolute position zero for axes 1 and 2).

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

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

The 6000 controller retains the absolute position, even while the unit is in the incremental mode. In steppers, the absolute position can be ascertained with the TPM and PM commands. In servos, use the TPC and PC commands.

Example

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


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Continuous Positioning Mode

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

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

You can manipulate the motor movement with either buffered or immediate commands. After you issue the GO command, buffered commands are not executed unless the continuous command execution mode (COMEXC1 command) is enabled. Once COMEXC1 is enabled, buffered commands are executed in the order in which they were programmed. More information on the COMEXC mode is provided on page 16.

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

Example A

```
COMEXC1          ; Enable continuous command processing mode
COMEXS1          ; Allow command execution to continue after stop
MC1              ; Sets axis 1 mode to continuous
A10              ; Sets axis 1 acceleration to 10
V1               ; Sets axis 1 velocity to 1
GO1              ; Initiates axis 1 move (Go)
WAIT(1VEL=1)     ; Wait to reach continuous velocity
T5               ; Time delay of 5 seconds
S1               ; Initiate stop of axis 1 move
WAIT(MOV=b0)     ; Wait for motion to completely stop on axis 1
COMEXC0          ; Disable continuous command processing mode
```

The motor accelerates to 1 unit/sec, continues at that rate for 5 seconds, and then decelerates to a stop.

Example B

```
DEF prog1        ; Begin definition of program prog1
COMEXC1          ; Enable continuous command processing mode
COMEXS1          ; Allow command execution to continue after stop
MC1              ; Set axis 1 to continuous positioning mode
A10              ; Set axis 1 acceleration to 10
V1               ; Set axis 1 velocity to 1
GO1              ; Initiate axis 1 move (Go)
WAIT(1VEL=1)     ; Wait for motor to reach continuous velocity
T3               ; Time delay of 3 seconds
A50              ; Set axis 1 acceleration to 50
V10              ; Set axis 1 velocity to 10
GO1              ; Initiate acceleration and velocity changes on axis 1
T5               ; Time delay of 5 seconds
S1               ; Initiate stop of axis 1 move
WAIT(MOV=b0)     ; Wait for motion to completely stop on axis 1
COMEXC0          ; Disable continuous command processing mode
END              ; End definition of program prog1
```

While in continuous mode, motion can be stopped if:

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

NOTE

While the axis is moving, you cannot change the parameters of some commands (such as ENC and HOM). This rule applies during the COMEXC1 mode and even if you prefix the command with an immediate command identifier (!). For more information, refer to *Restricted Commands During Motion* on page 18.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

End-of-Travel Limits

The 6000 controller can respond to both hardware and software end-of-travel limits. The purpose of end-of-travel limits is to prevent the motor's load from traveling past defined limits.

HARDWARE LIMITS

6000 controllers are shipped from the factory with the hardware end-of-travel limits enabled, but not connected. Therefore, motion will not be allowed until you do one of the following:

- Install limit switches or jumper the end-of-travel limit terminals to the GND terminal (*refer to your product's Installation Guide for wiring instructions*).
- Disable the limits with the LH command (recommended only if the load is not coupled).
- Reverse the active level of the limits with the LHLVL command.

Related Commands:

LH.....Hard limit enable
LHAD.....Hard limit decel
LHADA...Hard limit decel (s)
LHLVL...Limit switch polarity
LS.....Soft limit enable
LSAD.....Soft limit decel
LSADA...Soft limit decel (s)
LSNEG...Soft limit (negative)
LSPOS...Soft limit (positive)
TLIM.....Hard limit status
TASF.....Bits 15-18 indicate if hard or soft limit was encountered
TERF.....Bit 2: hard limit hit
Bit 3: soft limit hit
(must enable ERROR checking bits 2 &3)

Once a hardware or software limit is reached, the 6000 controller will decelerate that axis at the rate specified with the LHAD and LSAD commands, respectively. Servos also use the LHADA and LSADA commands for S-curve deceleration.

Typically, software and hardware limits are positioned in such a way that when the software limit is reached the motor/load will start to decelerate toward the hardware limit. This will allow for a much smoother stop at the hardware limit. Software limits can be used regardless of incremental or absolute positioning.

The default active level of the hardware limits is active-low, requiring normally-closed limit switches. If you wish to change the active level to active-high, use the LHLVL1 command.

Software limits are defined by the LSPOS and LSNEG commands. The LSPOS command establishes the limit in the positive direction, LSNEG for the negative direction limit. These limits are enabled with the LS command and are scaled by the SCLD command. The software limits are referenced from a position of absolute zero. Both software limits may be defined with positive values (e.g., axis #2 in the example below) or negative values. *Care must be taken when performing incremental moves because the software limits are always defined in absolute terms.* They must be large enough to accommodate the moves, or a new zero reference position must be defined (using the PSET command) before each move.

NOTES

- To ensure proper motion when using soft end-of-travel limits, be sure to set the LSPOS value to an absolute value greater than the LSNEG value.
- Stepper products: If your system is moving heavy loads or operating at high velocities, you may need to decrease the LHAD command value (deceleration rate) to prevent the motor from stalling (Zeta drives and ZETA610n may compensate without reducing decel).
- If you reverse the commanded direction polarity (CMDDIR1), you should swap the hardware end-of-travel switch connections to maintain a positive correlation with the commanded direction.

Example In this example, the hardware and software limits are enabled on axes #1 and #2, and disabled on axes #3 and #4. The distance scaling command (SCLD) is used to define software limit locations in revolutions from the absolute zero position (assumes a 4000 step/rev resolution). Deceleration rates are specified for both software and hardware limits. If a limit is encountered, the motors will decelerate to a stop.

```
@ERES4000    ; Set encoder resolution to 4000 steps/rev (all axes)
SCALE1       ; Enable scaling
@SCLD4000    ; Allow user to program soft limit distance in revs (all axes)
@SCLA4000    ; Program soft limit accel/decel in revs/sec/sec (all axes)
LH3,3,0,0    ; Enable limits 1 and 2, disable limits 3 and 4
LHAD10,10    ; Set hard limit deceleration
LSAD5,10     ; Set soft limit deceleration
LSNEG0,2     ; Set neg. direction soft limit (axis 1: 0 revs; axis2: 2 revs)
LSPOS10,20   ; Establish pos. soft limit (axis 1: 10 revs; axis 1: 20 revs)
LS3,3,0,0    ; Enable soft limits 1 and 2, disable limits 3 and 4
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Homing *(Using the Home Inputs)*

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

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

As soon as the homing operation is successfully completed, the absolute position register is reset to zero, thus establishing a zero reference position (this applies also to the voltage register if using ANI feedback).

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

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

Command	**Homing Function** (see respective command descriptions for further details)	**Default**
HOM	Initial the homing move. To start the homing move in the positive direction, use HOMØ; to home in the negative direction, use HOM1.	HOMx (do not home)
HOMA	Acceleration while homing.	HOMA1Ø (10 units/sec^2)
HOMAA	S-curve acceleration while homing (servos only).	HOMAA1Ø (10 units/sec^2)
HOMAD	Deceleration while homing.	HOMAD1Ø (10 units/sec^2)
HOMADA	S-curve deceleration while homing (servos only).	HOMADA1Ø (10 units/sec^2)
HOMBAC	Back up to home. The load will decelerate to a stop after encountering the active edge of the home region, and then will move in the opposite direction at the HOMVF velocity until the active edge of the home region is encountered. Allows the use of HOMEDG and HOMDF.	HOMBACØ (function disabled)
HOMDF	Final approach direction —during backup to home (HOMBAC) or during homing to the Z channel input of an incremental encoder (HOMZ).	HOMDFØ (positive direction)
HOMEDG	Specify the side of the home switch on which to stop (either the positive-travel side or the negative-travel side).	HOMEGDØ (positive-travel side of switch)
HOMLVL	Define the home limit input active level (i.e., the state, high or low, which is to be considered an activation of the input). To use a normally-open switch, select active low (HOMLVLØ); to use a normally-closed switch, select active high (HOMLVL1).	HOMLVLØ (active-low, use a normally-closed switch)
HOMV	Velocity while seeking the home position (see also HOMVF).	HOMV1 (1 unit/sec)
HOMVF	Velocity while in final approach to home position—during backup to home (HOMBAC) or during homing to the Z channel input of an incremental encoder (HOMZ).	HOMVF.1 (0.1 unit/sec)
HOMZ	Home to the Z channel input from an incremental encoder. NOTE: The home limit input must be active prior to homing to the Z channel. (This feature is not applicable to the OEM-AT6400.)	HOMZØ (function disabled)

AVOID PAUSE & RESUME DURING HOMING

Avoid using pause and resume functions during the homing operation. 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.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

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

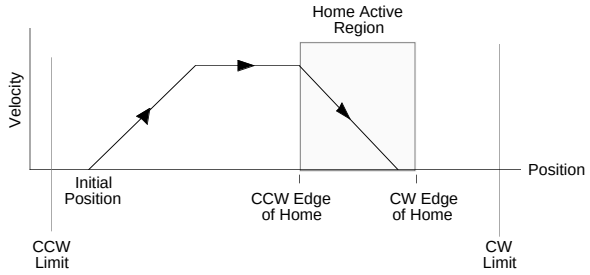


Figure A. Homing in a CW Direction (HOMØ) with backup to home disabled (HOMBACØ)

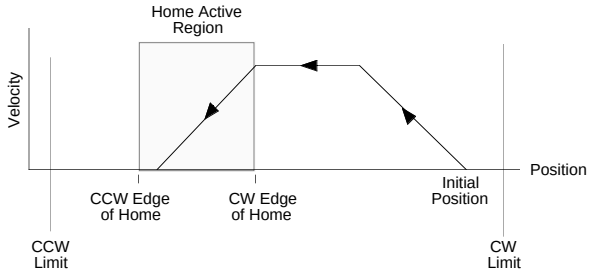


Figure B. Homing in a CCW Direction (HOM1) with backup to home disabled (HOMBACØ)

Positive Homing, Backup to Home Enabled

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

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

Step 1 A CW home move is started with the HOMØ command at the HOMA and HOMAA accelerations. Default HOMA is 10 revs (or volts or inches) per sec^2.

Step 2 The HOMV velocity is reached (move continues at that velocity until home input goes active).

Step 3 The CCW edge of the home input is detected, this means the home input is active. At this time the move is decelerated at the HOMAD and HOMADA command values. It does not matter if the home input becomes inactive during this deceleration.

Step 4 After stopping, the direction is reversed and a second move with a peak velocity specified by the HOMVF value is started.

Step 5 This move continues until the CCW edge of the home input is reached.

Step 6 Upon reaching the CCW edge, the move is decelerated at the HOMAD and HOMADA command values, the direction is reversed, and another move is started in the CW direction at the HOMVF velocity.

Step 7 As soon as the home input CW edge is reached, this last move is immediately terminated. The load is at home and the absolute position register is reset to zero.

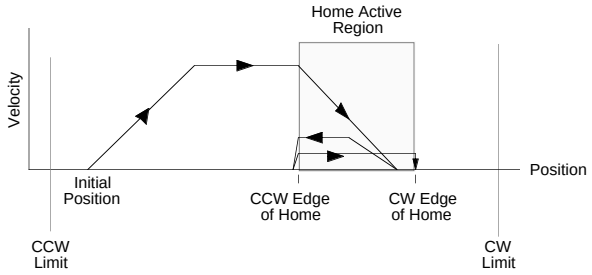


Figure C. Homing in a CW Direction (HOMØ) with HOMBAC1, HOMEDGØ, HOMDFØ

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

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

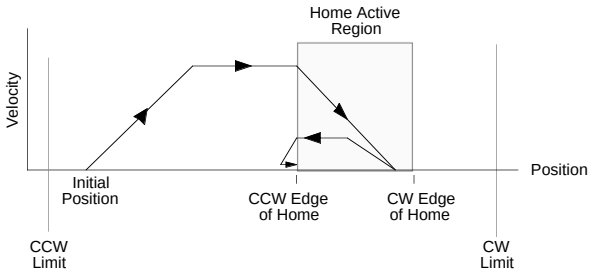


Figure D. Homing in a CW Direction (HOMØ) with HOMBAC1, HOMEDG1, HOMDFØ

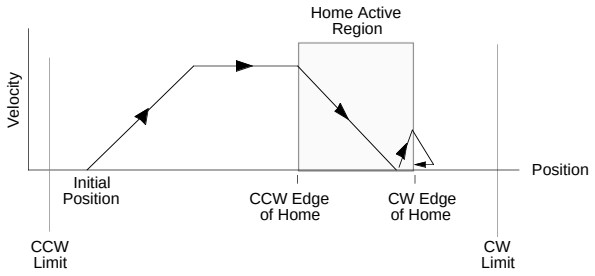


Figure E. Homing in a CW Direction (HOMØ) with HOMBAC1, HOMEDGØ, HOMDF1

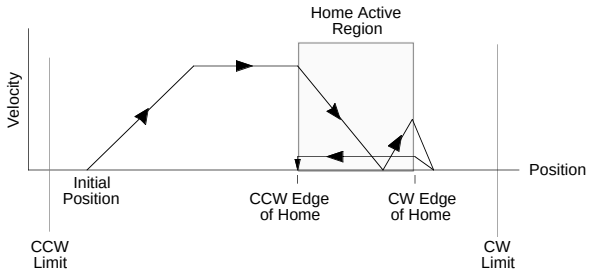


Figure F. Homing in a CW Direction (HOMØ) with HOMBAC1, HOMEDG1, HOMDF1

Negative Homing, Backup to Home Enabled

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

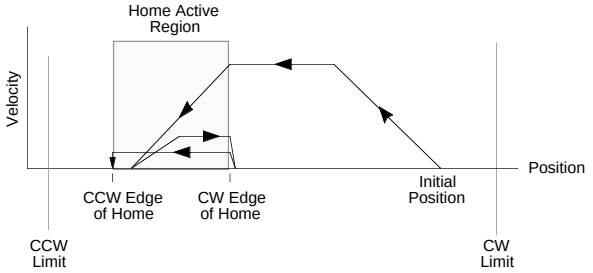


Figure G. Homing in a CCW Direction (HOM1) with HOMBAC1, HOMEDG1, HOMDF1

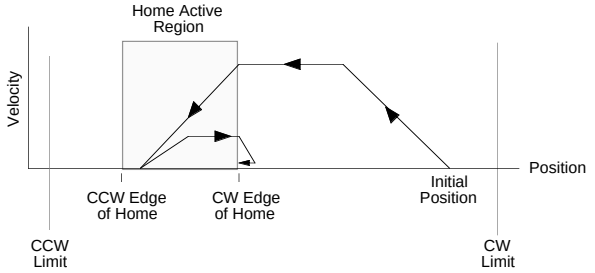


Figure H. Homing in a CCW Direction (HOM1) with HOMBAC1, HOMEDGØ, HOMDF1

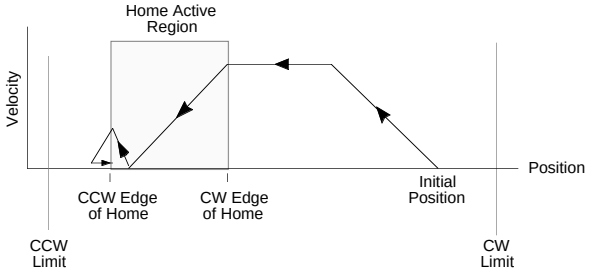


Figure I. Homing in a CCW Direction (HOM1) with HOMBAC1, HOMEDG1, HOMDFØ

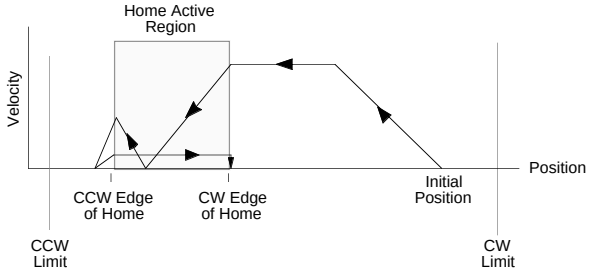


Figure J. Homing in a CCW Direction (HOM1) with HOMBAC1, HOMEDGØ, HOMDFØ

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Homing Using The Z-Channel

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

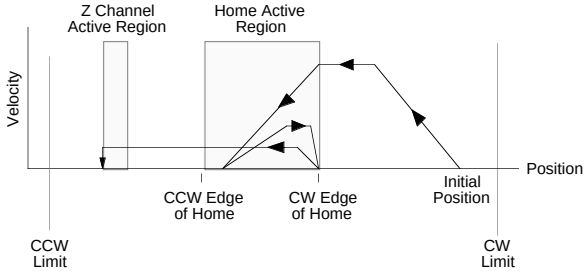


Figure K. Homing in a CCW Direction (`HOM1`) with `HOMBAC1`, `HOMEDG1`, `HOMDF1`

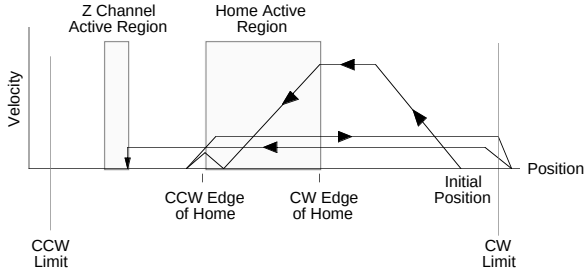


Figure L. Homing in a CCW Direction (`HOM1`) with `HOMBAC1`, `HOMEDGØ`, `HOMDFØ`

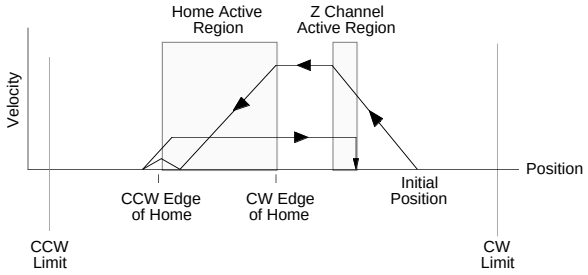


Figure M. Homing in a CCW Direction (`HOM1`) with `HOMBAC1`, `HOMEDGØ`, `HOMDFØ`

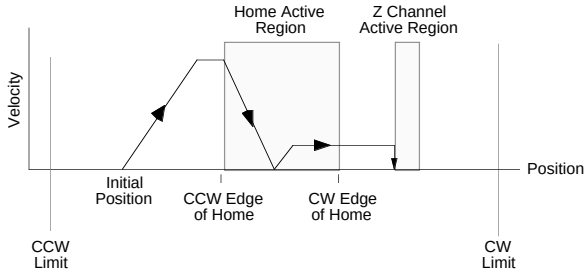


Figure N. Homing in a CW Direction (`HOMØ`) with `HOMBACØ`, `HOMEDGØ`, `HOMDFØ`

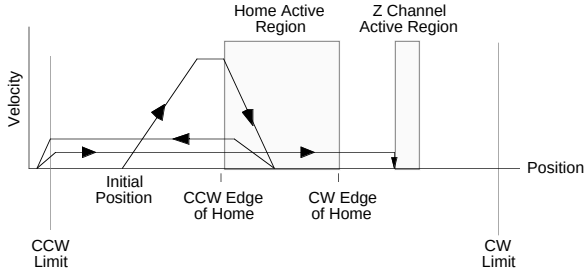


Figure O. Homing in a CW Direction (`HOMØ`) with `HOMBACØ`, `HOMEDGØ`, `HOMDF1`

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Closed-Loop Stepper Setup *(steppers only)*

NOTE

Closed Loop operation is not available for the OEM-AT6400.

Closed-loop refers to operating with position feedback from an encoder. This section discusses how you can use an incremental encoder to perform closed-loop functions with the controller. When using an encoder in an application, you may configure the following (described below):

- Encoder Resolution
- Encoder vs. Motor Step Mode
- Position Maintenance Enable & Correction Parameters
- Position Maintenance Deadband
- Stall Detection & Kill-on-Stall
- Stall Deadband
- Counter
- Encoder Polarity
- Commanded Direction Polarity

Encoder Resolution

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

Compumotor Encoder	Resolution	ERES Command
E Series	4000 counts/rev (1000 lines)	ERES4000
OEM-83 Series	4000 counts/rev (1000 lines)	ERES4000
OEM-57 Series	2048 counts/rev (512 lines)	ERES2048

IMPORTANT NOTE

To achieve accurate encoder step positioning, the drive resolution (DRES), see page 81, should be at least four times greater than the encoder resolution (ERES). This allows the motor to successfully find the commanded encoder step position.

Encoder Step Mode

The controller can perform moves in either motor steps or encoder steps. In Motor Step Mode (ENCØ), the distance command (D) defines moves in motor steps. Motor Step Mode is the controller's default setting. In Encoder Step mode (ENC1), the distance command defines moves in encoder steps.

Position Maintenance

The EPM1 command enables the position maintenance function. To enable position maintenance, you must first connect an encoder and enable the Encoder Step Mode (ENC1).

Enabling position maintenance causes the indexer to *servo* (adjust) the motor until the correct encoder position is achieved. This occurs at the end of a move (if the final position is incorrect) or any time the indexer senses a change in position while the motor is at zero velocity.

The EPMG and EPMV commands define the gain factor and the maximum velocity of a position maintenance correction move. The gain factor is the velocity increment used per encoder step of error. The EPMV command sets the maximum velocity a position maintenance move can achieve. If either of these is too large, position instability may result. If either is too small, position correction may be too slow. *The values are determined by experiment.*

Position Maintenance Deadband

The EPMDB command sets the number of encoder steps of error allowed before a position maintenance correction move is initiated. This is useful for situations in which it is not desirable to have the load position continuously corrected, but correction is required outside the deadband. Positioning under a microscope is an example.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Stall Detection & Kill-on-Stall

The ESTALL1 command allows the controller to detect stall conditions. If used with *Kill-on-Stall* enabled (ESK1 command), the move in progress will be aborted upon detecting a stall. If queried with the ER or the AS commands, the user may branch to any other section of program when a stall is detected. Refer to the ENC, ER, and AS command descriptions in the ***6000 Series Software Reference*** for more information.

Stall Detection functions in either motor step or encoder step mode. Kill-on-Stall functions only if the stall detection is enabled (ESTALL1).

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

Stall Deadband

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

Encoder Set Up Example

The example below illustrates the features discussed in the previous paragraphs. The first statement defines the motor resolutions. The next statement defines the number of encoder steps per encoder revolution. Standard 1000-line encoders are used on all axes that produce 4000 quadrature steps/rev. Stall detect and kill-on-stall are enabled. If a stall is detected, the motor's movement is killed. The next group of commands define the deadband width and enable the position maintenance function. The controller will attempt to move to the correct encoder position until it is within the specified deadband.

Examples

```
DRES25000,25000,25000,200      ; Set drive resolution
ERES4000,4000,4000,4000        ; Set encoder resolution
ESK1111                        ; Enable kill motion on stall
ESDB0,0,10,10                  ; Set stall deadband
ESTALL1111                     ; Enable stall detection
EPMG1000,1000,1000,10001       ; Set position maintenance gain
EPMV1,1,1,1                    ; Set position maintenance velocity
EPM1111                        ; Enable position maintenance
ENC1111                        ; Enable encoder step mode
```

Use the Encoder as a Counter

Each encoder channel can be redefined as a 16-bit up/down counter. The CNTE command redefines the encoder channels. 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-, indicates a positive count direction. A negative differential signal, when measured between B+ and B-, indicates a negative 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, issue the CNTR command or apply a positive differential signal to Z+ and Z-. The counter will be reset on the rising edge. Allow 50µs for the counter to reset, and allow 50µs after reset for counting to begin.

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

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Encoder Polarity

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

Immediately after issuing the ENCPOL command, the encoder will start counting in the opposite direction (including all encoder position registers). The polarity is immediately changed whether or not encoder step mode is enabled (ENC1).

NOTES

- Changing the feedback polarity effectively invalidates any existing offset position (PSET) setting; therefore, you will have to re-establish the PSET position.
- The ENCPOL command is automatically saved in non-volatile RAM (stand-alone products).
- If you wish to reverse the commanded direction of motion, first make sure there is a direct correlation between commanded direction and encoder direction, then issue the appropriate CMDDIR command to reverse both the commanded direction and the encoder direction (see CMDDIR command description for full details).

Programming Scenario (as seen in a terminal emulator)

This programming scenario assumes the encoder polarity is reversed from the commanded direction (e.g., commanding a move of +10 units, yields and encoder position of -10 units).

```
>  PSET0        ; Define current position of axis 1 as position zero
>  1TPE         ; Check the position of encoder #1
*1TPE+0         (response indicates encoder #1 is at position zero)

>  MA0          ; Select incremental positioning mode
>  D+8000       ; Set distance to 8,000 units in the positive direction
>  GO1          ; Move axis 1 a distance of 8,000 units
>  1TPE         ; Check the position of encoder #1
*1TPE-8000      (response shows that encoder #1 is at position -8000,
                 the minus sign indicates that the encoder is counting in the wrong direction)

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

Commanded Direction Polarity

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

NOTES

- **The CMDDIR command cannot be executed while motion is in progress or while the drive is enabled**. For example, you could wait for motion to be complete (indicated when TAS and AS bit #1 is a zero) and then use the DRIVE command to disable the appropriate axis before executing the CMDDIR command.
- Before changing the commanded direction polarity, make sure there is a direct correlation between the commanded direction and the direction of the encoder counts (i.e., a positive commanded direction from the controller must result in positive counts from the encoder).
- Once you change the commanded direction polarity, you should swap the end-of-travel limit connections to maintain a positive correlation with the commanded direction.
- The CMDDIR setting is automatically saved in non-volatile memory (stand-alone products).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Servo Setup *(servo products only)*

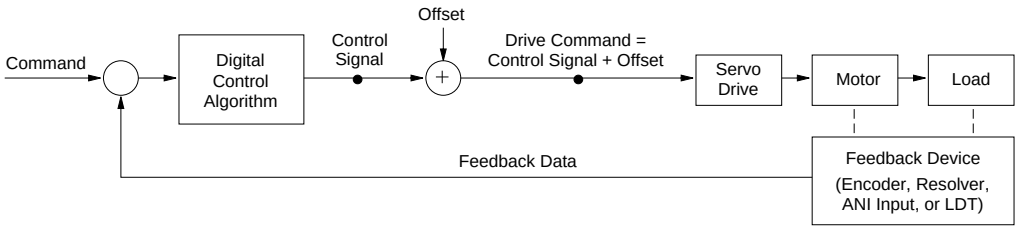
EXAMPLES

Pages 103 and 104 show examples of servo setup commands as used in a setup program.

Servo products are *closed loop* systems (see diagram below). As such, they require configuration so that the control algorithm can effectively calculate the value of the control signal output based on the feedback data from a feedback device, such as an encoder, resolver, ANI input, or LDT:

- Tuning:
  - Select gains to achieve optimal servo performance.
  - Select the participating axes (as on page 79) and the trajectory/servo ratio. These two parameters determine the performance for the *servo sampling update*, the *motion trajectory update*, and the *system update*.
  - Define the maximum position error, which is the maximum allowable error between the commanded position and the position as measured by the feedback device.
  - Select and configure the feedback device(s) you are using with your controller.
- Feedback Device Polarity (optional): Servo stability requires a direct correlation between the commanded direction and the direction of the feedback device counts (i.e., a positive commanded direction from the controller must result in positive counts from the feedback device). If the feedback device is counting in the wrong direction, you may reverse the polarity without having to change the actual wiring to the feedback device.
- Commanded Direction Polarity (optional): may be required 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.
- Dither (optional): Add a square wave signal to the analog output to overcome *stiction*.
- DAC Output Limits (optional): Define the region of the controller's -10V to +10V output range to be used for your application.
- Servo Control Signal Offset (optional): During setup and troubleshooting, you may need to disable the servo control algorithm and directly control the analog output to the drive.

Servo System: Closed Loop



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Tuning

For an overview of servo control terminology and techniques, refer to the Servo Tuner User Guide*, or to your product's installation guide.*

To assure optimum performance and , you should tune your servo system. The goal of the tuning process is to define the gain settings, servo performance, and feedback setup (see command list below) that you can incorporate into your application program. (Typically, these commands are placed into a setup program – see examples on page 103).

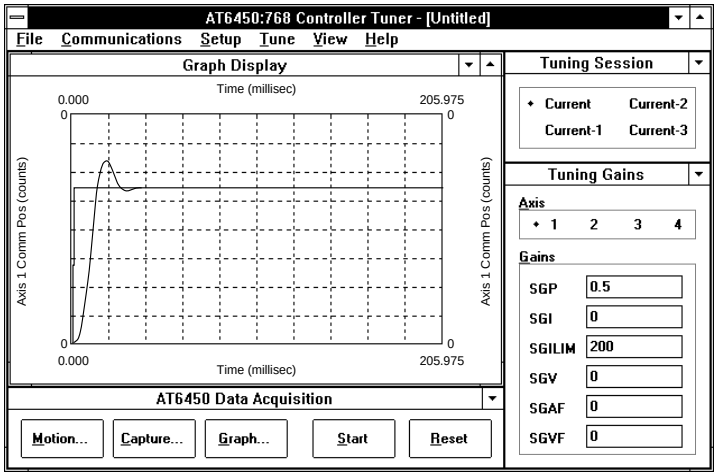
We recommend using Servo Tuner™, a graphical tuning software tool (see note below). If you prefer a more empirical method, refer to the procedures in your 6000 product's installation guide.

Servo Tuning Software Available

To effectively tune your 6000 servo controller (and any velocity drives you may be using), use the interactive tuning features in the Servo Tuner™. It greatly improves your efficiency and gives you powerful graphical tools to measure the performance of the system.

Servo Tuner is included as an integral element of Motion Builder™, an optional icon-based programming tool. Servo Tuner is also available as an optional add-on module to Motion Architect (it does not automatically come with the basic Motion Architect software package). Instructions for using Servo Tuner are provided in the ***Servo Tuner User Guide*** and in Motion Builder's online Help system and ***Motion Builder Startup Guide & Tutorial***.

To order Motion Builder or the Servo Tuner add-on module to Motion Architect, contact your local Automation Technology Center (ATC) or distributor.



Tuning-Related Commands (see ***6000 Series Software Reference*** or the ***Servo Tuner User Guide*** for details)

Tuning Gains:

SGP * ..... Sets the proportional gain in the **P**IV&F servo algorithm.

SGI * ..... Sets the integral gain in the P**I**V&F servo algorithm.

SGV * ..... Sets the velocity gain in the PI**V**&F servo algorithm.

SGAF * ..... Sets the acceleration feedforward gain in the PIV&$\mathbf{F_a}$ algorithm.

SGVF * ..... Sets the velocity feedforward gain in the PIV&$\mathbf{F_v}$ algorithm.

SGILIM ..... Sets a limit on the correctional control signal that results from the integral gain action trying to compensate for a position error that persists too long.

SGENB ..... Enables a previously-saved set of PIV&F gains. A set of gains (specific to the current feedback source selected with the SFB command) is saved using the SGSET command.

SGSET ..... Saves the presently-defined set of PIV&F gains as a *gain set* (specific to the current feedback source on each axis). Up to 5 gain sets can be saved and enabled at any point in a move profile, allowing different gains at different points in the profile.

* 6270 only: Negative gains (SGPN, SGIN, SGVN, SGAFN, SGVFN) are automatically applied when the position error is negative.

Servo Performance:

INDAX ..... Selects the number of available axes to use (see page 79).

SSFR ..... Sets the ratio between the update rate of the move trajectory and the update rate of the servo action. Affects the *servo sampling update*, the *motion trajectory update*, and the *system update*.

Feedback Setup:

SFB ..... Selects the servo feedback device. Options (depending on the product) are: encoder, resolver, ANI input, or LDT. IMPORTANT: Parameters for scaling, tuning gains, max. position error (SMPER), and position offset (PSET) are specific to the feedback device selected (with the SFB command) at the time the parameters are entered (see programming examples on page 103).

ERES ..... Encoder resolution (also selects the resolution for a resolver).

LDTRES ..... (6270 only) Sets the LDT resolution.

LDTGRD ..... (6270 only) Sets the LDT gradient.

LDTUPD ..... (6270 only) Selects the rate at which the LDT position is sampled.

SMPER ..... Sets the maximum allowable error between the commanded position and the actual position as measured by the feedback device. If the error exceeds this limit, the controller activates the Shutdown output and sets the DAC output to zero (plus any S0FFS offset). If there is no offset, the motor will freewheel to a stop. You can enable the ERR0R command to continually check for this error condition (ERROR.12-1), and when it occurs to branch to a programmed response defined in the ERR0RP program.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Feedback Device Polarity

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

To change the commanded and feedback polarity, use the CMDDIR command instead (see page 101 for details).

If the feedback device is counting in the wrong direction, you may reverse the polarity with the respective polarity reversal command (ENCPOL, ANIPOL or LDTPOL). This allows you to reverse the counting direction without having to change the actual wiring to the feedback device. For example, if the encoder on axis 2 counted in the wrong direction, you could issue the ENCPOLx1 command to correct the polarity. The feedback devices available vary by product (see table below):

	Feedback Devices (& associated polarity command)		
Product	**Encoder (ENCPOL)**	**ANI Input (ANIPOL)**	**LDT (LDTPOL)**
AT6n50 and 625n	*command not available*	*command not available*	*not applicable*
615n	Yes *	Optional	*not applicable*
6270	Yes	Optional	Yes

* 615n: The ENCPOL command affects the external encoder, not the internal resolver.

Immediately after issuing the ENCPOL, ANIPOL, or LDTPOL command, the respective feedback device will start counting in the opposite direction (including all feedback device position registers). The polarity of the respective feedback device is immediately changed whether or not the specific device is currently selected with the SFB command.

NOTES

- You can not change the feedback polarity on a specific axis while that axis is moving.
- Changing the feedback polarity effectively invalidates any existing offset position (PSET) setting; therefore, you will have to re-establish the PSET position.
- The ENCPOL, ANIPOL and LDTPOL commands are 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 feedback device direction, then issue the appropriate CMDDIR command to reverse both the commanded direction and the feedback device direction (see CMDDIR command description or page 101 for full details).

Programming Scenario (as seen in a terminal emulator)

```
> SFB1          ;Select encoder feedback for axis 1
> SMPER100      ;Set maximum position error to 100 units on axis 1
> PSET0         ;Define current position of axis 1 as position zero
> 1TPE          ;Check the position of encoder #1
*1TPE+0         (response indicates encoder #1 is at position zero)

> MA0           ;Select incremental positioning mode
> D+8000        ;Set distance to 8,000 units in the positive direction
> GO1           ;Move axis 1. If the encoder polarity is incorrect, the axis will be unstable
                ;and will stop (drive disabled) as soon as the maximum position error of
                ;100 units is reached.
> 1TPE          ;Check the position of encoder #1
*1TPE-100       (response should show that encoder #1 is approximately at position -100;
                the minus sign indicates that the encoder is counting in the wrong direction)

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


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Commanded Direction Polarity

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

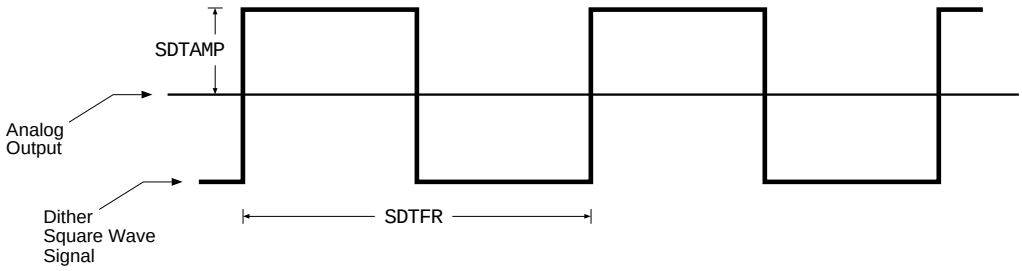
The CMDDIR command allows you to reverse the direction that the controller considers to be the "positive" direction; this also reverses the polarity of the counts from the feedback devices. Thus, using the CMDDIR command, you can reverse the referenced direction of motion without the need to (a) change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

NOTES

- **The CMDDIR command cannot be executed while motion is in progress or while the drive/valve is enabled.** For example, you could wait for motion to be complete (indicated when TAS and AS bit #1 is a zero) and then use the DRIVE command to disable the appropriate axis before executing the CMDDIR command.
- Before changing the commanded direction polarity, make sure there is a direct correlation between the commanded direction and the direction of the feedback source counts (i.e., a positive commanded direction from the controller must result in positive counts from the feedback device). 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 is not implemented in the AT6n50 and 625n products.

Dither

Dither is a square-wave signal added to the analog output and is used to keep the motor or the hydraulic valve moving slightly for the purpose of reducing stiction (see illustration below). Two commands are used to select the amplitude and frequency of the dither signal—SDTAMP and SDTFR.



SSFR (servo sampling frequency) / SDTFR = Dither Frequency (cycles/sec)

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.

☞ Refer to the SSFR command description or the *Servo Tuner User Guide* for a discussion on the servo sampling rate.

The actual dither frequency is determined by the ratio of the servo sampling frequency (SSFR & INDAX settings) and the SDTFR value. As an example, if the SSFR value is 4 and the INDAX value is 2, the servo sampling rate is 3571 samples per second. Then, at SSFR4, an SDTFR value of 46 (default setting) would yield a 77 Hz dither frequency (3571/46 = 77). 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.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

DAC Output Limits

If you will not be using the entire -10V to +10V range of the 6000 controller's analog output, you can set up maximum (DACLIM) and minimum (DACMIN) limits. (DACMIN is available only for the 6270.)

6270 EXAMPLE: 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).

```
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)
```

Servo Control Signal Offset

The SOFFS command (and SOFFSN command, 6270 only) provides a means of setting the controller's analog output to a known voltage value. This could be useful for these occasions:

- Testing motion in an open-loop configuration (all gains set to zero).
- If the commanded output is set to zero (motor/valve is supposed to be stationary), but it keeps moving, you can impose an offset value to stop motion. *This is the same effect as the balance input on most analog servo drives.*
- 6270: Selecting the mid-range voltage for a valve (see example in *DAC Output Limits* above).

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

CAUTION — Torque Drive Users

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

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Servo Setup Examples

This section shows examples of how the servo setup commands might be incorporated into a setup program. More information on creating and executing setup programs is provided on page 14.

6250 & AT6250:

```
DEF SETUP       ; Begin definition of SETUP program for AT6250 controller
DRIVE00         ; Disable both drives
INDAX2          ; Place both axes in use
SSFR4           ; Select servo sampling frequency ratio
DRFLVL11        ; Set drive fault level to active high
KDRIVE11        ; Invoke the Disable Drive On Kill feature for both axes

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

;***********************
;  setup for ANI       *
;***********************
SFB2,2          ; Select ANI feedback for both axes (subsequent scaling,
                ; gains, ;SMPER, and PSET parameters are specific to ANI
                ; feedback)
SCLA819,819     ; Set scaling for programming accel/decel in volts/sec/sec
SCLV819,819     ; Set scaling for programming velocity in volts/sec
SCLD819,819     ; Set scaling for programming distances in volts
SGP1,1          ; Set proportional feedback gain
SGI0,0          ; Set integral feedback gain
SGV.5,.5        ; Set velocity feedback gain
SMPER.01,.01    ; Set max. position error to 1/100 of a volt (8 ANI counts)
PSET5,5         ; Set current position as absolute position 5

SFB1,1          ; Re-select encoder feedback for start of main program
END             ; End definition of SETUP program

;*************************************************************************
;* 6250: Enter the STARTP SETUP command to assign SETUP as the startup   *
;*       program to be automatically executed on power up or RESET.      *
;*                                                                       *
;* AT6250: Download & execute the SETUP program (preps for subsequent    *
;*         operations) before running the main program.                  *
;*************************************************************************
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

6270:

```
DEF PWRUP       ; Begin definition of PWRUP program
DRIVE00         ; Disable both valves/drives
INDAX2          ; Place both axes in use
SSFR4           ; Select servo sampling frequency ratio
DRFLVL11        ; Set drive fault level to active high
KDRIVE11        ; Enable the DISABLE ON KILL feature for both axes

;*************************************************************************
;* setup for LDT                                                         *
;* (will need to switch between LDT, encoder, and ANI feedback)          *
;*************************************************************************
SFB3,3          ; Select LDT feedback for both axes (subsequent scaling,
                ; gains, servo offset, SMPER, and PSET parameters are
                ; specific to LDT feedback)
LDTGRD9,9       ; Set LDT gradient to 9µs/inch for both LDTs
LDTRES432,432   ; Set LDT resolution to 432 counts/inch
LDTUPD1,1       ; Set LDT position update rate equal to the system update
                ; rate (see table in SSFR command description)
SCLA432,432     ; Set scaling for programming accel/decel in inches/sec/sec
SCLV432,432     ; Set scaling for programming velocity in inches/sec
SCLD432,432     ; Set scaling for programming distances in inches
SGP50,50        ; Set proportional feedback gain
SGI.2,.2        ; Set integral feedback gain
SGV30,30        ; Set velocity feedback gain
SMPER.01,.01    ; Set max. position error to 1/100 of an inch
                ; (4 LDT counts)
PSET10,10       ; Set current position as absolute position 10

;***********************
;  setup for encoder   *
;***********************
SFB1            ; Select encoder feedback for axis 1 (subsequent scaling,
                ; gains, servo offset, SMPER, and PSET parameters are
                ; specific to encoder feedback)
ERES4000        ; Set encoder resolution to 4,000 counts/rev
SCLA4000        ; Set scaling for programming accel/decel in revs/sec/sec
SCLV4000        ; Set scaling for programming velocity in revs/sec
SCLD4000        ; Set scaling for programming distances in revs
SGP.5           ; Set proportional feedback gain
SGI1            ; Set integral feedback gain
SGV1            ; Set velocity feedback gain
SMPER.001,.001  ; Set maximum position error to 1/1000 of a rev
                ; (4 encoder counts)
PSET0           ; Set current position as absolute position zero

;***********************
;  setup for ANI       *
;***********************
SFB2,2          ; Select ANI feedback for both axes (subsequent scaling,
                ; gains, servo offset, SMPER, and PSET parameters are
                ; specific to ANI feedback)
SCLA819,819     ; Set scaling for programming accel/decel in volts/sec/sec
SCLV819,819     ; Set scaling for programming velocity in volts/sec
SCLD819,819     ; Set scaling for programming distances in volts
SGP1,1          ; Set proportional feedback gain
SGI0,0          ; Set integral feedback gain
SGV.5,.5        ; Set velocity feedback gain
SMPER.01,.01    ; Set maximum position error to 1/100 of a volt
                ; (8 ANI counts)
PSET5,5         ; Set current position as absolute position 5

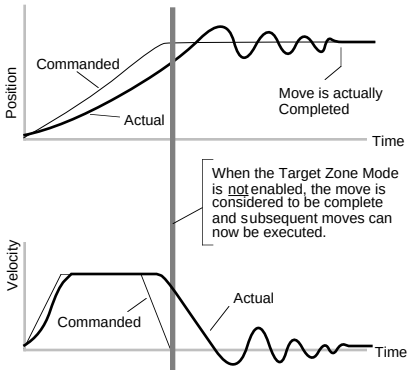
SFB3,3          ; Select LDT feedback for start of main program
END             ; End definition of PWRUP program

STARTP PWRUP    ; Assign PWRUP as the startup program to be automatically
                ; executed on power up or RESET
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Target Zone Mode *(Move Completion Criteria)*

Under default operation (Target Zone Mode not enabled), the 6000 product's move completion criteria is simply derived from the move trajectory. The 6000 product considers the current preset move to be complete when the commanded trajectory has reached the desired target position; after that, subsequent commands/moves can be executed for that same axis. Consequently, the next move or external operation can begin before the actual position has settled to the commanded position (see diagram).



NOTE
Stepper products may use the Target Zone mode ONLY if operating in encoder step mode (ENC1) with encoder feedback.

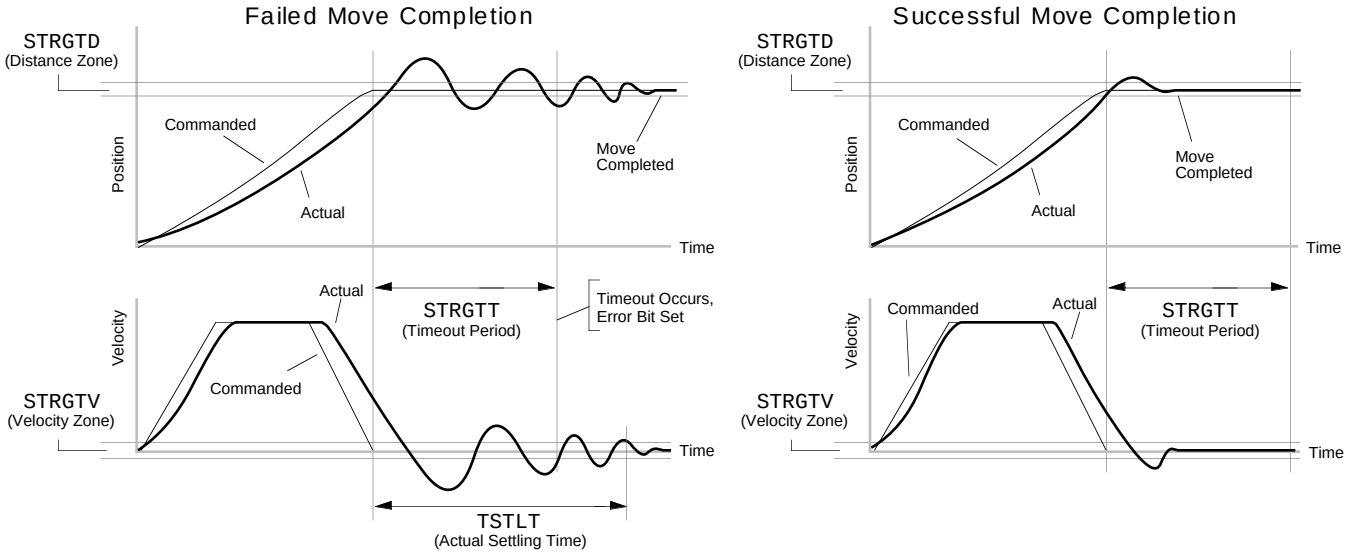
To prevent premature command execution before the actual position settles into the commanded position, use the *Target Zone Mode*. In this mode, enabled with the STRGTE command, the move cannot be considered complete until the actual position and actual velocity are within the *target zone* (that is, within the distance zone defined by STRGTD and less than or equal to the velocity defined by STRGTV). If the load does not settle into the target zone before the timeout period set with the STRGTT command, the 6000 product detects a *timeout error* (see illustration below).

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

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

Damping is critical

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



Checking the Settling Time

Checking the Actual Settling Time: Using the TSTLT command, you can display the actual time it took the last move to settle into the target zone (that is, within the distance zone defined by STRGTD and less than or equal to the velocity defined by STRGTV). The reported value represents milliseconds. **This command is usable whether or not the Target Zone Settling Mode is enabled with the STRGTE command.**

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Programmable Inputs and Outputs *(including triggers and auxiliary outputs)*

Programmable inputs and outputs allow the controller to detect and respond to the state of switches, thumbwheels, electronic sensors, and outputs of other equipment such as drives and PLCs. Based on the state of the inputs and outputs, read with the [IN] and [OUT] commands, the controller can make program flow decisions and assign values to binary variables for subsequent mathematical operations. These operations and the associated program flow, branching, and variable commands are listed below.

Operation based on I/O State	Associated Commands	See Also*
I/O state assigned to a binary variable	[IN], [OUT], VARB	*Variables* (Chapter 1, page 22)
I/O state used as a basis for comparison in conditional branching & looping statements	[IN], [OUT], IF, ELSE, NIF, REPEAT, UNTIL, WAIT, WHILE, NWHILE	*Program Flow Control* (Chapter 1, page 25)
Input state used as a basis for a conditional GO	[IN], GOWHEN	*Synchronizing Motion* (Chapter 5, page 187)
I/O state used as a basis for a program interrupt (GOSUB) conditional statement	ONIN	*Program Interrupts* (Chapter 1, page 29)

* Refer also to the respective command descriptions in the ***6000 Series Software Reference***.

CHECKING I/O STATUS: As discussed below, you can program and check the status of each input and output with the INFNC and OUTFNC commands, respectively. To receive a binary report of the state (on or off) of the I/O, use the TIN command (inputs) or the TOUT command (outputs); the reports are in the format are identified in the I/O bit pattern table below.

DEFINE I/O ACTIVE LEVELS: Many people refer to a voltage level when referencing the state of programmable inputs and outputs. Using the INLVL and OUTLVL commands, you can define the logic levels of all the programmable inputs and outputs (general-purpose I/O, plus trigger inputs and auxiliary outputs) as positive or negative. The product defaults to an input/output level of Ø volts as its active level (referred to as “active low”); thus, a “1” will appear in a status command referencing an input/output state when the voltage level is Ø volts (e.g., if the device driving input #1 on axis 1 is on (sinking current) the 1TIN.1 status command will report *1).

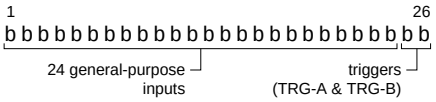
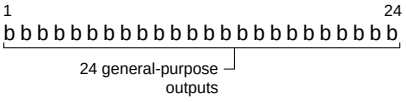
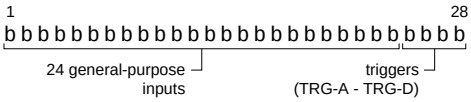
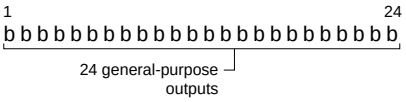
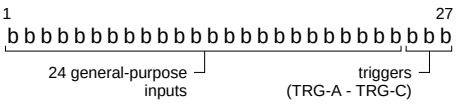
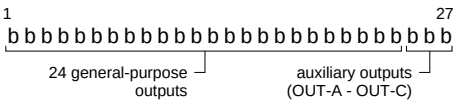
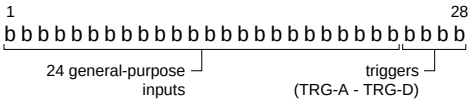
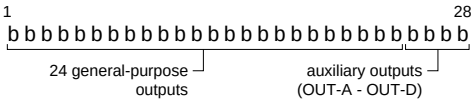
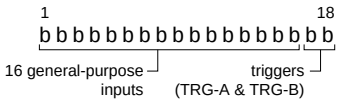
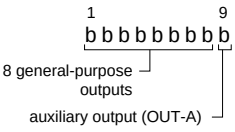
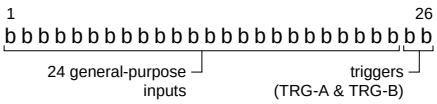
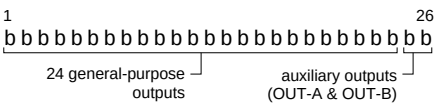
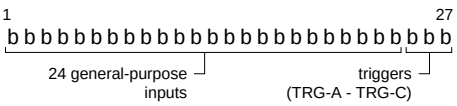
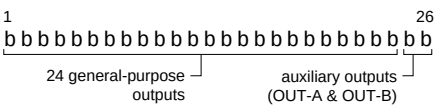
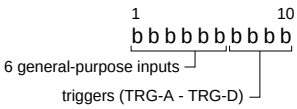
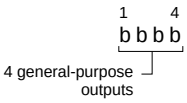
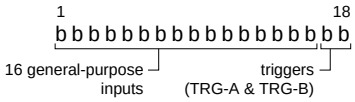
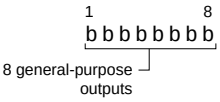
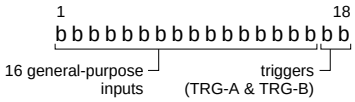
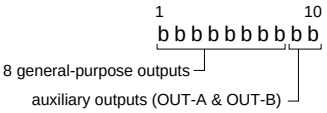
I/O UPDATE RATE: The programmable inputs and outputs are sampled at the “system update rate.” The update rate for steppers is 2 ms. The update rate for servos is determined by the INDAX and SSFR command settings (see table in SSFR command description).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Programmable I/O Bit Patterns

I/O specifications are provided in your product's installation guide.

The total number of 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.

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

Servo Products Only:

* `INEN` command has no effect on trigger inputs when they are configured as *Trigger Interrupt* inputs with the `INFNCi-H` command.

** `OUTEN` command has no effect on auxiliary outputs when they are configured as *Output-on-Position* outputs with the `OUTFNCi-H` command.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Input Functions

The input functions are assigned with the INFNCi-<a>c command. The "i" represents the number of the input bit (see bit pattern table on page 107). The "<a>" represents the number of the axis, if required. The "c" represents the letter designator of the function, A through Q (see list below). For example, the INFNC5-2D command configures output #5 to function as a *stop* input, stopping motion on axis #2 when activated.

> **NOTE**
> To activate the function of an input with the INFNC command, you must first enable the input functions with the INFEN1 command. Because the INFEN1 command also enables the drive fault input, you should verify the fault active level (DRFLVL) is set properly.

Letter Designator	Function
A	No Function (**default**)
B	BCD Program Select
C	Kill
D	Stop
E	Pause/Continue
F	User Fault
G	<RESERVED>
H	Trigger Interrupt for position capture or registration (trigger inputs only) *
I	Interrupt to PC-AT (bus-based controllers only)
J	Jog+ (positive direction)
K	Jog- (negative direction)
L	Jog Speed Select
M-O	<RESERVED>
P	Program Select
Q	Program Security

* Special trigger functions can be assigned with the TRGFN command (see page __ for details).

Input Status

As shown below, you can use the INFNC command to define and check the current function and state (on or off) of one or all the inputs. The TIN command also reports the inputs' state, but in a binary format (see bit pattern table on page 107). The TSTAT command also reports the inputs' functions (by letter designator) and their current state (see page 232).

To use the state of the inputs as a basis for conditional statements (IF, REPEAT, WHILE, GOWHEN, etc.), use the IN assignment/comparison operator (refer to the *Conditional Looping and Branching* section on page 24, and *Synchronizing Motion* on page 186).

Code Examples

Command — **Description**

INFNC ..... Query status of all inputs; response indicating default conditions is:
```
*INFNC1-A NO FUNCTION INPUT - STATUS OFF
*INFNC2-A NO FUNCTION INPUT - STATUS OFF
```
(response continues until all programmable inputs are reported)

INFNC1 ..... Query status of input #1; response indicating default conditions is:
```
*INFNC1-A NO FUNCTION INPUT - STATUS OFF
```

INFNC1-D ..... Change input #1 to function as a Stop input

INFNC1 ..... Query status of input #1; response should be now be:
```
*INFNC1-D STOP INPUT - STATUS OFF
```

TIN ..... Query binary status report of all inputs;
response indicating default conditions for AT6400 is:
```
*TIN0000_0000_0000_0000_0000_0000_0000
```

IF(IN.8=b1) ..... IF statement that evaluates true when input #8 is active

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Input Debounce Time

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 2 trigger inputs.

General-Purpose Input Debounce: The input debounce time for the 24 general-purpose inputs is the period of time that the input must be held in a certain state before the controller recognizes it. This directly affects the rate at which the inputs can change state and be recognized.

Trigger 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.

> **Triggers: Non-debounced state used for conditional statements**
>
> If you use the ONIN command or if you use the status of programmable inputs (IN comparison operator) as the condition for an IF, GOWHEN, or WAIT statement (e.g., IF(IN.3=b1)), it is the non-debounced state that is recognized. Therefore, rapid transitions as fast as one system update period (2 ms for steppers, see SSFR table for servos) will be noticed by these statements.

The INDEB command syntax is INDEB<i>,<i>. The first <i> is the input number (see page 107 for input bit assignments) and the second <i> is the debounce time **in even increments** of milliseconds (ms). The debounce time range is 1-250 ms. The default debounce time is 4 ms for the general-purpose inputs, and 24 ms (servos) or 50 ms (steppers) for the trigger inputs.

If the first <i> is in the range corresponding to general-purpose inputs (e.g., input numbers 1-24 for the AT6400), the specified debounce time is assigned to all general-purpose inputs. If the first <i> is in the range corresponding to the trigger inputs (e.g., input numbers 25-28 for the AT6400), the specified debounce is assigned only to the specified trigger input. For example (AT6400), the INDEB5,6 command assigns a debounce time of 6 ms to all 24 general-purpose inputs. The INDEB26,12 command assigns a debounce time of 12 ms only to input #26, which is trigger B (**TRG-B**).

No Function (INFNCi-A)

When an input is defined as a *No Function* input (default function), the input is used as a standard input. You can then use this input to synchronize or trigger program events.

Example

```
DEL prog1        ; Precaution: Delete a program before defining it
DEF prog1        ; Begin definition of program prog1
INFEN1           ; Enable input functions
INFNC1-A         ; No function for input 1
INFNC2-A         ; No function for input 2
INFNC3-D         ; Input 3 is a stop input
A10              ; Set acceleration
V10              ; Set velocity
D5               ; Set distance
WAIT(IN=b1XX)    ; Wait for input 1
GO1              ; Initiate motion
IF(IN=bX1)       ; If input 2
1TFB             ; Transfer feedback device position for axis 1
NIF              ; End IF statement
END              ; End definition of program prog1
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

BCD Program Select
(INFNCi-B)

General-purpose inputs (*not trigger inputs*) can be defined as *BCD program select* inputs. This allows you to execute defined programs (DEF command) by activating the program select inputs. Program select inputs are assigned BCD weights. The table to the right shows the BCD weights of the controller's inputs when inputs 1-8 are configured as program select inputs. The inputs are weighted with the least weight on the smallest numbered input.

Input #	BCD Weight
Input 1	1
Input 2	2
Input 3	4
Input 4	8
Input 5	10
Input 6	20
Input 7	40
Input 8	80

If inputs 6, 9, 10 and 13 are selected instead of inputs 5, 6, 7 and 8, then the weights would be as follows:

Input #6 = 10
Input #9 = 20
Input #10 = 40
Input #13 = 80

If, for example you defined 100 programs, a maximum of 8 inputs are required to select all possible programs.

The program number is determined by the order in which the program was downloaded to the controller. The program number can be obtained through the TDIR command (see programming example below) — *The number in front of each program name is the BCD weight you need to achieve in order to execute the program.*

If the inputs are configured as in the above table, activating inputs 2 and 3 will execute program #6. Activating inputs 1, 4, and 6 will execute program #29.

To execute programs using the program select lines, enable the INSELP command. Once enabled, the controller will continuously scan the input lines and execute the program selected by the active program select lines. To disable scanning for program select inputs, enter !INSELPØ or place INSELPØ in a program that can be selected.

Once enabled (INSELP1), the controller will run the program number that the active program select inputs and their respective BCD weights represent. After executing and completing the selected program, the controller will scan the inputs again. If a program is selected that has not been defined, no program will be executed.

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

Example

```
RESET       ; Return controller to power-up conditions
ERASE       ; Erase all programs
DEF PROG1   ; Begin definition of program PROG1
TPE         ; Transfer position of encoders
END         ; End program
DEF PROG2   ; Begin definition of program PROG2
TREV        ; Transfer software revision
END         ; End program
DEF PROG3   ; Begin definition of program PROG3
TSTAT       ; Transfer statistics
END         ; End program
INFNC1-B    ; Assign input 1 as a BCD program select input
INFNC2-B    ; Assign input 2 as a BCD program select input
INFEN1      ; Enable input functions
INSELP1,50  ; Enable scanning inputs, levels must be maintained for 50ms
TDIR        ; Display number and name of programs stored in memory
; response from TDIR should be similar to following:
;                   *1 - PROG1 USES 6 BYTES
;                   *2 - PROG2 USES 18 BYTES
;                   *3 - PROG3 USES 99 BYTES
;                   *32877 OF 33000 BYTES (98%) PROGRAM MEMORY REMAINING
;                   *500 OF 500 SEGMENTS (100%) COMPILED MEMORY REMAINING
```

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

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

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

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Kill
(INFNCi-C)

An input defined as a *Kill* input will stop motion on all axes, the program currently in progress will also be terminated, the commands currently in the command buffer will be eliminated, and the drive will be left in the enabled state (DRIVE1111).

Servos: Motion is stopped at the rate set with the hard limit (LHAD& LHADA) commands.

Steppers: Motion is stopped with no deceleration ramp (**CAUTION**: Because there is no deceleration ramp, the motor may stall, or the drive may fault, possibly causing the load to free wheel—no control from the drive).

If the kill input goes active, and if error-checking bit 6 is set (ERRORxxxxx1), the error status will be reported by bit 6 of the TERF, TER and ER commands and the error program (ERRORP) will be executed to respond to the error condition (see page 30 for more on error programming).

Disabling the Drive on a Kill (Servos Only)

If your application requires you to *disable* (*shut down* or *de-energize*) the drive in a Kill situation, set the controller to the *Disable Drive on Kill* mode with the KDRIVE1 command. In this mode, a kill command or kill input will shut down the valve or drive immediately. If you are using a drive, the shutdown outputs are activated and the motor/load will then be allowed to *free wheel* (without control from the drive) to a stop. If you are using a hydraulic valve, the kill will set the command output to zero, causing the valve to return immediately to the neutral (*null*) position and hold the load at that position. To re-enable the valve or drive, issue the DRIVE1111 command to that axis.

Stop
(INFNCi-D)

An input defined as a *Stop* input will stop motion on any one or all axes. Deceleration is controlled by the programmed AD/ADA deceleration ramp. After the Stop input is received, further program execution is dependent upon the COMEXS command setting:

If error-checking bit #8 is enabled (e.g., ERROR.8-1), then a stop input will cause a branch to the error program (for more information, see *Error Handling* on page 30).

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

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

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

COMEXS2: Upon receiving a stop input, motion will stop, 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-aP commands, to continue.

Pause/Continue
(INFNCi-E)

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

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

COMEXR1: Upon receiving a pause input, both motion and program execution will be paused; the motion stop function is used to halt motion. Releasing the pause input or issuing a !C command will resume motion and program execution. *You cannot resume program execution while the move in progress is decelerating.*

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

User Fault
(INFNCi-F)

An input defined as a *User Fault* input acts as an immediate Kill (!K) command, stopping motion on all axes and terminating program execution. If error-checking bit #7 is enabled (e.g., ERROR.7-1), then a user fault input will cause a branch to the ERRORP error program (for more information, see *Error Handling* on page 30) and the occurrence of a user fault input will be reported by error bit #7 (see TERF, TER and ER commands).

Servos: Motion is stopped at the rate set with the hard limit (LHAD& LHADA) commands.

Steppers: Motion is stopped with no deceleration ramp (**CAUTION**: Because there is no deceleration ramp, the motor may stall, or the drive may fault, possibly causing the load to *free wheel*—no control from the drive).

Trigger Interrupt
(INFNCi-H)

Any trigger input may be defined as a Trigger Interrupt input and can be used for these functions:

- Position Capture (see below)
- Special trigger functions assigned with the TRGFN command (see below)
- Registration (see discussion on page 182)

NOTE:
When configured as Trigger Interrupts, the triggers cannot be affected by the input enable (INEN) command.

The Trigger Interrupt function can be assigned only to the trigger inputs (not to the general-purpose inputs). For example, if your controller has 4 trigger inputs represented by input numbers 25-28 (input bit assignments vary by product – see page 107), to assign the interrupt function to all 4 triggers, you would program the inputs as follows:

- Positions captured only by trigger A: assign function with INFNC25-H
- Positions captured only by trigger B: assign function with INFNC26-H
- Positions captured only by trigger C: assign function with INFNC27-H
- Positions captured only by trigger D: assign function with INFNC28-H

Trigger Input Debounce

The trigger interrupt input is debounced for 24 ms (servos) or 50 ms (steppers) before another input on the same trigger is recognized. If your application requires a shorter debounce time, you can change it with the INDEB command (refer to *Input Debounce Time* on page 109).

Position Capture

Certain applications (such as coordinate measurement machines) require latching the current position upon receiving an input. When a trigger input defined as a trigger interrupt input is activated, the commanded position (*motor position* for steppers) and the position of all feedback devices on all axes are captured at one time. The position information is stored in registers and is available at the next system update through the use of transfer and assignment/comparison commands (see table below).

Captured Information	Transfer Command	Assignment/Comparison Command
Motor Position (steppers only)	TPCM	PCM
Commanded Position (servos only)	TPCC	PCC
Encoder Position *	TPCE	PCE
Resolver Position **	TPCEA	PCEA
ANI Position (servos with ANI only)	TPCA	PCA
ANI Voltage (servos with ANI only)	TCA	CA
LDT Position	TPCL	PCL

* 615n: Encoder position is captured by trigger B and reported with the TPCEB and PCEB commands.
** 615n: Resolver position is captured by trigger A and reported with the TPCEA and PCEA commands.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Position Capture Accuracy

Servos: If you are capturing the position/value of a feedback source (encoder, ANI, LDT) that is currently selected with the SFB command, the position capture accuracy is ±50 µs * current velocity.

If you are capturing the position of a device that is not selected with the SFB command, the last sampled position is simply stored as the captured position.

***Exceptions*:**

- The captured commanded position is always interpolated from the last sampled position (of the feedback device selected with the SFB command) and the position error, and the time elapsed since the last sample.
- Regardless of the SFB selection, one encoder position is latched in hardware within ±1 encoder count (at max. encoder frequency) after its *dedicated* trigger input is activated (see table below). Triggers A-D are dedicated to encoders 1-4, respectively. **This encoder capture method offers the highest accuracy.**

Encoder	AT6250	AT6450	615n *	625n	6270	OEM6250
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.

Steppers: There is a time delay of up to 50 µs between activating the trigger interrupt input and capturing the position; therefore, the accuracy of the captured position is equal to 50 µs multiplied by the velocity of the axis at the time the input was activated.

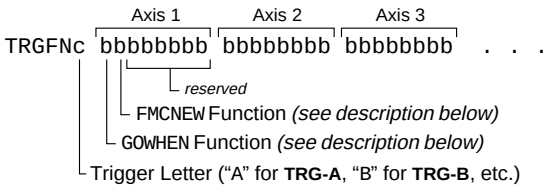
System status bits #25 through #28, reported with the TSSF, TSS and SS commands, are set to 1 when the positions have been captured on trigger inputs A through D, respectively. As soon as the captured information is transferred or assigned/compared (see command list above), the respective system status bit is cleared, but the information is still available from the register until it is overwritten by a new latch from the trigger input.

Captured values are offset by the PSET command. If scaling is enabled (SCALE1), the captured values are scaled by the programmed SCLD value.

Trigger Functions

The Trigger Functions command (TRGFN) allows you to assign additional functions to trigger inputs that have been defined as trigger interrupt inputs (INFNCi-H):

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 ("1" enables the function, "Ø" disables the function, "x" leaves the function bit unchanged):



GOWHEN Function: (TRGFNc1xxxxxx) 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 page 186 or to the GOWHEN command description for additional details. Axis status bit #26 (reported with TASF, TAS, or AS) is set to one (1) when there is a pending GOWHEN condition initiated by a TRGFNc1xxxxxx command; this bit is cleared when the trigger is activated or when a stop or kill command is issued.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

FMCNEW Function: (TRGFNCx1xxxxx) Allows a new Following master cycle to begin when the specified trigger input (C) goes active. For additional details on master cycles, refer to page 208 or to the FMCNEW command description.

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.

Sample 6000 Code

```
INFNC26-H           ; Assign trigger B (TRG-B) on the AT6400 (input #26) to
                    ; function as a trigger interrupt input.
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.
```

Registration

If registration is enabled (with the RE command), activating a trigger interrupt input will initiate registration move(s) defined with the REG command.

Refer to page 182 for details on the Registration feature.

Interrupt to PC AT (INFNCi-I)

Bus-based Products Only

An input specified as an *Interrupt to PC -AT* input will interrupt the PC-AT when the input goes active, if bit 27 of the hardware interrupt enable (INTHW) command is enabled. For the interrupt to be received by the PC-AT itself, one of the 8 interrupt DIP switches (S2) on the controller card must be enabled. Once the input has been activated, the interrupt will activate.

☞ *DIP switch settings are provided in the product's installation guide.*

It is the responsibility of the corresponding interrupt service routine to determine the cause of the interrupt. The cause of the interrupt can be determined by reading the interrupt status from the fast status area (see page 43), or by using the TINT command. If the cause of the interrupt was an input, bit 27 of the interrupt status will be set.

Jogging the Motor

(INFNCi-aJ)
(INFNCi-aK)
(INFNCi-aL)

In some applications, you may want to manually move (*jog*) the load. The jog mode is enabled with the JOG command. You can configure the jog functions with these INFNC commands:

- INFNCi-aJ..... Jog in the positive counting direction when the input is active, stop when the input is inactive.
- INFNCi-aK..... Jog in the negative counting direction when the input is active, stop when the input is inactive.
- INFNCi-aL..... Select the high (JOGVH) or low (JOGVL) velocity setting for jog motion. Activating the input selects high velocity, deactivating the input selects low velocity.

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

- JOGVH..... High velocity range for jogging. The high velocity is used when the jogging speed-select input (configured with INFNCi-aL) is active.
- JOGVL..... Low velocity range for jogging. The low velocity is used when the jogging speed-select input (configured with INFNCi-aL) is inactive.
- JOGA..... Jog acceleration
- JOGAA..... Jog acceleration (s-curve profile)
- JOGAD..... Jog deceleration
- JOGADA..... Jog deceleration (s-curve profile)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

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

Example *Step 1* **Define program for jog setup:**

```
DEF prog1     ; Begin definition of program prog1
SCALE0        ; Disable scaling
JOGA25        ; Set jog acceleration to 25
JOGAD25       ; Set jog deceleration to 25
JOGVL.5       ; Sets low-speed  jog velocity to 0.5
JOGVH5        ; Sets high-speed  jog velocity to 5
INFEN1        ; Enable input functions
INFNC1-1J     ; Sets input 1 as a positive-direction jog input
INFNC2-1K     ; Sets input 2 as a negative-direction jog input
INFNC3-1L     ; Sets input 3 as a speed-select input
JOG1          ; Enable Jog function for axis 1
END           ; End program definition
```

Step 2 Download and run the prog1 program.

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

Step 4 Activate input 2 to move the load in the negative direction at a velocity of 0.5 units/sec (until input 2 is released).

Step 5 Activate input 3 to switch to high-speed jogging.

Step 6 Repeat steps 3 and 4 to perform high-speed jogging at the JOGVH value (5 rps).

One-to-One Program Select (INFNCi-iP)

Inputs can be defined as *One-to-One Program Select* inputs (INFNCi-iP). This allows programs defined by the DEF command to be executed by activating an input. Different from BCD Program Select inputs, One-to-One Program Select inputs correspond directly to a specific program number. The program number is determined by the order in which the program was downloaded to the controller. The program number can be obtained with the TDIR command—the number noted before the program name is to be used in the second variable of the INFNCi-iP definition (see programming example below).

To execute programs using the program select lines, enable one-to-one program selection (INSELP2). Once enabled, the controller will continuously scan the input lines and execute the program selected by the active program select line. To disable scanning of the program select lines, enter !INSELPØ, or place INSELPØ in a program that can be selected.

Example

```
RESET         ; Return controller to power-up default conditions
DEF proga     ; Begin definition of program proga
TFB           ; Transfer position of feedback devices
END           ; End program
DEF progb     ; Begin definition of program progb
TREV          ; Transfer software revision
END           ; End program
DEF progc     ; Begin definition of program progc
TSTAT         ; Transfer statistics
END           ; End program
TDIR          ; Response should show:   *1 - PROGA USES 36 BYTES
              ;                         *2 - PROGB USES 70 BYTES
              ;                         *3 - PROGC USES 133 BYTES
INFNC4-1P     ; Input 4 will select proga
INFNC5-2P     ; Input 5 will select progb
INFNC6-3P     ; Input 6 will select progc
INFEN1        ; Enable input functions
INSELP2,50    ; Enable scanning of inputs with a strobe time of 50 ms
```

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

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

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Program Security (INFNCi-Q)

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.

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 the DEF, DEL, ERASE, MEMORY, or INFNC commands without the use of the program access input, you must issue the INFENØ command to disable programmable input functions, make the desired command changes, and then issue the INFEN1 command to re-enable the programmable input functions.

Stand-alone products: The INFNCi-Q command is not saved in battery-backed RAM, so you may want to put it in the start-up program (STARTP).

Output Functions

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

The output functions are assigned with the OUTFNCi-<a>c command. The "i" represents the number of the output bit (see bit pattern table on page 107). The "<a>" represents the number of the axis and is optional for the B, D, E, and G functions (see list below); when no axis specifier is given, the output will be activated when the condition occurs on any axis. The "c" represents the letter designator of the function (A through H). For example, the OUTFNC5-2D command configures general-purpose output #5 to activate when axis #2 encounters a hard or soft limit.

NOTE

To activate the function of an output with the OUTFNC command, you must enable the output functions with the OUTFEN1 command.

Letter Designator	Function	Servo Products	Stepper Products
A	Programmable Output (**default function**)	✓	✓
B	Moving/Not Moving (or In Position)	✓	✓
C	Program in Progress	✓	✓
D	End-of-Travel Limit Encountered	✓	✓
E	Stall Indicator	n/a	✓
F	Fault Indicator (indicates drive fault input or user fault input is active)	✓	✓
G	Position Error Exceeds Max. Limit	✓	n/a
H	Output on Position (auxiliary outputs only)	✓	n/a

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Output Status

As shown below, you can use the OUTFNC command to determine the current function and state (on or off) of one or all the outputs. The TOUT command also reports the outputs' state, but in a binary format (see bit pattern table on page 107). The TSTAT command also reports the outputs' functions (by letter designator) and their current state (see page 232).

Code Examples

Command **Response**

OUTFNC.....Query the status of all outputs:
*OUTFNC1-A PROGRAMMABLE OUTPUT - STATUS OFF
*OUTFNC2-F FAULT OUTPUT - STATUS OFF
(response continues until all programmable outputs are reported)

OUTFNC1.....Query the assigned function and current status of output #1:
*OUTFNC1-A PROGRAMMABLE OUTPUT - STATUS OFF

OUTFNC1-C.....Change output #1 to function as a *Program in Progress* output

OUTFNC1.....Query the status of output #1; response should be now be:
*OUTFNC1-C PROGRAM IN PROGRESS OUTPUT - STATUS OFF

TOUT.....Query binary status report of all outputs (for AT6400):
*TOUT0000_0000_0000_0000_0000_0000_0000_0000

Programmable Output
(OUTFNCi-A)

The default function for the outputs is *Programmable*. As such, the output is used as a standard output, turning it on or off with the OUT or OUTALL commands to affect processes external to the controller. To view the state of the outputs, use the TOUT command. To use the state of the outputs as a basis for conditional branching or looping statements (IF, REPEAT, WHILE, etc.), use the [OUT] command (refer to the *Conditional Looping and Branching* section, page 25, for details).

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

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

If the target zone mode is enabled (STRGTE1), the output will not change state until the move completion criteria set with the STRGTD and STRGTV commands has been met. (For more information, refer to the *Target Zone* section on page 105, or refer to the ***Servo Tuner User Guide***.) In this manner, the Moving/Not Moving output functions as an *In Position* output.

Example

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

```
SCALE0          ; Disable scaling
MC0             ; Sets axis 1 to preset positioning mode
MA0             ; Select incremental positioning mode
A10             ; Sets axis 1 acceleration to 10
V5              ; Sets axis 1 velocity to 5
D4000           ; Sets axis 1 distance to 4,000 steps
OUTFEN1         ; Enable output functions
OUTFNC1-A       ; Sets output 1 as a programmable output
OUTFNC2-A       ; Sets output 2 as a programmable output
OUTFNC3-1B      ; Sets output 3 as a axis 1 Moving/Not Moving output
OUT10           ; Turns output 1 on and output 2 off
GO1             ; Initiates axis 1 move
OUT01           ; Turns output 1 off and output 2 on
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Program in Progress
(OUTFNCi-C)

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

Limit Encountered
(OUTFNCi-<a>D)

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

If a hard or soft limit is encountered, you will not be able to move the motor/load in that same direction until you clear the limit by changing direction (D) and issuing a GO command. (An alternative is to disable the limits with the LHØ command, but this is recommended only if the motor is not coupled to the load.) The event of encountering an end-of-travel limit is also reported with axis status bits 15-18 (see TASF, TAS and AS commands, summary on page 233).

Stall Indicator
(OUTFNCi-<a>E)
Steppers Only
(n/a to OEM-AT6400)

When assigned the *Stall Indicator* function, the output will activate when a stall is detected. To detect a stall, you must first connect an encoder, 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. Refer to *Closed-Loop Stepper Setup* on page 95 for further discussion on stall detection.

Fault Output
(OUTFNCi-F)

When assigned the *Fault Output* function, the output will activate when either the user fault input or the drive fault input becomes active.

The user fault input is a general-purpose input defined as a user fault input with the INFNCi-F command (see page 112).

For the AT6n00, AT6n50, 625n, and 6200, the drive fault input is found on the **DRIVE** connector, pin #5. Make sure the drive fault active level (DRFLVL) is appropriate for the drive you are using, and the input functions are enabled (INFEN1). For the packaged drive/controller products (e.g., 6104, 6152, 6201) the drive fault input is connected internally and the default DRFLVL and INFEN1 command values are appropriate to the internal drive(s). *The drive fault input is not available on the OEM-AT6400.*

Maximum Position Error Exceeded
(OUTFNCi-<a>G)
Servos Only

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

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

NOTE

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

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Output on Position (OUTFNCi-H) ***Servos Only***

The *Output on Position* functions are assigned to the axes and corresponding auxiliary outputs:

- For axis 1: Function assigned only to **OUT-A** (e.g., `OUTFNC25-H` command for the AT6n50)
- For axis 2: Function assigned only to **OUT-B** (e.g., `OUTFNC26-H` command for the AT6n50)
- For axis 3: Function assigned only to **OUT-C** (e.g., `OUTFNC27-H` command for the AT6n50)
- For axis 4: Function assigned only to **OUT-D** (e.g., `OUTFNC28-H` command for the AT6450)

Not Available For ANI Feedback

The Output On Position feature can be used only with encoder or LDT feedback and is not operational with ANI feedback.

The *Output on Position* parameters are configured with the `OUTPA`, `OUTPB`, `OUTPC`, and `OUTPD` commands:

1st data field (`b`): `1` enables the *output on position* function; `Ø` disables the function. If a subsequent `SFB` command is executed, the function is disabled.

2nd data field (`b`): `1` sets the position comparison in the 3rd data field (`r`) to an incremental position;
`Ø` sets the position comparison in the 3rd data field (`r`) to an absolute position.

3rd data field (`r`): Represents the scalable distance with which the actual (encoder) position is to be compared (distance is either incremental or absolute, depending on the setting of the 2nd data field).

4th data field (`i`): Represents the time (in milliseconds) the output is to stay active. If this data field is set to `Ø`, the output will stay active for as long as the actual distance equals or exceeds the distance specified in the 3rd data field. (This is valid only for the absolute mode—2nd data field set to `Ø`).

If an incremental distance is used for comparison (2nd data field set to `1`), the output activates when the actual position is greater than or equal to the specified distance, and stays active for the specified time.

If an absolute distance is used for comparison (2nd data field set to `Ø`), the output activates when the actual position is greater than or equal to the specified absolute distance, and stays active for the specified time.

Output On Position — Only During Motion

The output activates only during motion; thus, issuing a `PSET` command to set the absolute position counter to activate the output on position will not turn on the output until the next motion occurs.

Sample Code for Setup

```
OUTFEN1              ; Enable programmable output functions
OUTFNC25-H           ; Define OUT-A (output #25 on AT6n50) as output on
                     ; position output, axis 1
OUTFNC26-H           ; Define OUT-B (output #26 on AT6n50) as output on
                     ; position output, axis 2
OUTPA1,0,+40000,50   ; Turn on OUT-A for 50 ms when the actual position
                     ; is greater than or equal to absolute position +40,000
OUTPB1,0,+24000,200  ; Turn on OUT-B for 200 ms when the actual position
                     ; is greater than or equal to absolute position +24,000
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Variable Arrays (*teaching* variable data)

More on variables: see page 18.

Variable data arrays provide a method of storing (*teaching*) variable data and later using the stored data as a source for motion program parameters. The variable data can be any value that can be stored in a numeric (VAR) variable (e.g., position, acceleration, velocity, etc). The variable data is stored into a *data program*, which is an array of *data elements* that have a specific address from which to write and read the variable data. Data programs do not contain 6000 Series commands.

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

Basics of Teach-Data Applications

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

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

Initialize a Data Program

This is accomplished with the DATSIZ command. 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). The second integer represents the total number of data elements (up to 6,500) you want in the data program. Upon issuing the DATSIZ command, the data program is created with all the data elements initialized with a value of zero.

The data program has a tabular structure, where the data elements are stored 4 to a line. Each line of data elements is called a *data statement*. Each element is numbered in sequential order from left to right (1 - 4) and top to bottom (1 - 4, 5 - 8, 9 - 12, etc.). You can use the TPROG 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=+0.0,+0.0,+0.0,+0.0
*DATA=+0.0,+0.0,+0.0,+0.0
*DATA=+0.0,+0.0,+0.0,+0.0
*DATA=+0.0
```

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

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Teach the Data to the Data Program

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

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

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

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

As an example, suppose data program #1 is configured to hold 13 data elements (DATSIZ1,13), the data pointer is configured to start at data element #1 and increment 1 data element after every value stored from the DATTCH command (DATPTR1,1,1), and the values of numeric variables #1 through #3 are already assigned (VAR1=2, VAR2=4, VAR3=8). If you then enter the DATTCH1,2,3 command, the values of VAR1 through VAR3 will be assigned respectively to the first three data elements in the data program, leaving the pointer pointing to data element #4. The response to the TPROG DATP1 command would be as follows (the text is highlighted to illustrate the final location of the data pointer after the DATTCH1,2,3 command is executed):

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

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

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

Recall the Data from the Data Program

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

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Summary of Related 6000 Series Commands

A detailed description of each command is provided in the **6000 Series Software Reference**.

`DATSIZ` ..... Establishes the number of data elements a specific data program is to contain. A new `DATPi` program name is automatically generated according to the number of the data program (`i` = 1 through 50). The memory required for the data program is subtracted from the memory allocated for user programs (see `MEMORY` command).

`DATPTR` ..... Moves the data pointer to a specific data element in any data program. This command also establishes the number of data elements by which the pointer increments after writing each data element from the `DATTCH` command and after recalling each data element with the `DAT` command.

`DATTCH` ..... Stores the variable data into the data program specified with the last `DATSIZ` or `DATPTR` command. After the data is stored, the data pointer is incremented the number of times entered in the third integer of the `DATPTR` command. *The data must first be assigned to a numeric variable before it can be taught to the data program.*

`TDPTR` ..... Responds with a 3-integer status report (`i,i,i`): First integer is the number of the active data program (the program # specified with the last `DATSIZ` or `DATPTR` command); Second integer is the location number of the data element to which the data pointer is currently pointing; Third integer is the increment set with the last `DATPTR` command.

`[ DPTR ]` .... From the currently active data program, uses the number of the data pointer's location in a numeric variable assignment operation or a conditional statement operation.

`[ DATPi ]` .. The name of the data program created after issuing the `DATSIZ` command. The integer (`i`) represents the number of the data program. Data programs can be deleted just like any other user program (e.g., `DEL DATP1`).

`[ DATi ]` .... From the data program specified with `i`, assigns the numeric value of the data element (currently pointed to by the data pointer) to a specified variable parameter in a 6000 series command (e.g., `D(DAT3),(DAT3)`).

Teach-Data Application Example

In this example, 2 axes of a 6000 Series controller are used to move a 2-axis stage. This example illustrates a common method of teaching a path by using the joystick to move the load into position, teach the position (triggered by the Joystick Release input), then move to the next position. Five positions will be taught from each axis (2 axes at one trigger), for a total of 10 data elements in the data program. After all 10 positions are taught to the data program, the controller will automatically move both axes to a home position, move to each position that was taught, and then return to the home position.

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

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

The joystick operation in this example is based on setting the Joystick Axes Select input (pin #15 on the Joystick connector) to high to select analog input channels #1 and #2 (pins #1 and #2) for joystick use, and using the Joystick Release input (pin #17) to trigger the position teach operation.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Step 1 **Initialize a Data Program.**

```
DEL DATP1        ; Delete data program #1 (DATP1) in preparation for
                 ; creating a new data program #1

DATSIZ1,1Ø       ; Create data program #1 (named DATP1) with an allocation
                 ; of 10 data elements. Each element is initialized to zero.
```

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

```
DEF SETUP        ; Begin definition of the subroutine called SETUP

JOYVH3,3         ; Set the high velocity speed to 3 units/sec

JOYVL.2,.2       ; Set the low velocity to 0.2 units/sec

JOYAXH1,2        ; When axes select input is set high, apply analog
                 ; input 1 to axis 1
                 ; and apply analog input 2 to axis 2

VAR1=Ø           ; Set variable #1 equal to zero

VAR2=Ø           ; Set variable #2 equal to zero

DRIVE11          ; Enable the drives for both axes

MA11             ; Enable the absolute positioning mode for both axes

END              ; End definition of the subroutine called SETUP
```

Step 3 **Define the TEACH Subroutine.**

```
DEF TEACH        ; Begin definition of the subroutine called TEACH

HOM11            ; Home both axes (absolute position counter is set to
                 ; zero after the homing move)

DATPTR1,1,1      ; Select data program #1 (DATP1) as the current active
                 ; data program, and move the data pointer to the first
                 ; data element. After each DATTCH value is stored to
                 ; DATP1, increment the data pointer by 1 data element.

REPEAT           ; Set up a repeat/until loop

JOY11            ; Enable joystick mode on both axes. At this point,
                 ; you can start moving the axes into position with the
                 ; joystick. While using the joystick, command processing
                 ; is stopped here until you activate the joystick
                 ; release input. Activating the joystick release input
                 ; disables the joystick mode and allows the subsequent
                 ; commands to be executed (assign the current positions
                 ; to the variables and then store the positions in the
                 ; data program).

VAR1=1PM         ; Set variable #1 equal to the position of motor 1

VAR2=2PM         ; Set variable #2 equal to the position of motor 2

DATTCH1,2        ; Store variable #1 and variable #2 into consecutive
                 ; data elements. (The first time through the repeat/until
                 ; loop, variable #1 is stored into data element #1 and
                 ; variable #2 is stored into data element #2. The data
                 ; pointer is automatically incremented once after each
                 ; data element and ends up pointing to the third data
                 ; element in anticipation of the next DATTCH command.)

WAIT(INO.5=b1)   ; Wait for the joystick release input to be de-activated

UNTIL(DPTR=1)    ; Repeat the loop until the data pointer wraps around
                 ; to data element #1 (data program full)

END              ; End definition of the subroutine called TEACH
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Step 4 **Define the `DOPATH` Subroutine.**

```
DEF DOPATH      ; Begin definition of the subroutine called DOPATH

HOM11           ; Move both axes to the home position
                ; (absolute counters set to zero)

A5Ø,5Ø          ; Set up the acceleration

V3,3            ; Set up the velocity

DATPTR1,1,1     ; Select data program #1 (DATP1) as the current active data
                ; program, and set the data pointer to the first data
                ; element. Increment the data pointer one element after
                ; every data assignment with the DAT command. If you wanted
                ; to move only axis 1 down the taught path, you would set
                ; the increment (third integer) to a 2, thus accessing only
                ; the axis 1 stored positions.

REPEAT          ; Set up a repeat/until loop

D(DAT1),(DAT1)  ; The position of axis 1 and axis 2 are recalled into
                ; the distance command

GO11            ; Move to the position

T.5             ; Wait for 0.5 seconds

UNTIL(DPTR=1)   ; Repeat the loop until the data pointer wraps around
                ; to data element #1 (all data elements have been read)

HOM11           ; Move both axes back to the home position

END             ; End definition of the subroutine called DOPATH
```

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

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

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

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

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

CHAPTER FOUR

4 User Interface Options

IN THIS CHAPTER

This chapter explains how to use these user interface options in your application:

- Safety Features....126
- I/O Device Interface (thumbwheels, PLCs, etc.)....128
- RP240 Remote Operator Panel (stand-alone products only)....130
- Joystick Interface (including feedrate override)....138
- ANI Analog Input Interface (servos with ANI option only)....142
- Auxiliary analog output for half-axis (servos with ANA output only)....142
- Host Computer Interface....143
- Graphical User Interface (GUI) development tools....144

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

> **⚠ WARNING ⚠**
>
> The 6000 Product is used to control your system's electrical and mechanical components. Therefore, you should test your system for safety under all potential conditions. Failure to do so can result in damage to equipment and/or serious injury to personnel.

To help ensure a safe operating environment, you should take advantage of the safety features listed below. These features must not be construed as the only methods of ensuring safety. *See Also* refers you to where you can find more in-depth information about the feature (system connections and/or programming instructions).

Feature	Description	See Also
Pulse Cut-off Input (**steppers only**)	The pulse-cut input (**P-CUT**), found on the product's screw-terminal connectors, is provided as an emergency stop input to the controller. *The P-CUT input is not available on the OEM-AT6400 product.* When you open the **P-CUT** input, with respect to **GND**, the step pulses being send out to the drives on all axes are immediately cut off—*This occurs independent of the microprocessor.*	Product's *Installation Guide*
Enable Input (**servos only**)	The enable input (**ENBL**), found on the product's screw-terminal connectors, is provided as an emergency stop input to the controller. When you open the **ENBL** input, with respect to **GND**, the analog output voltage between **CMD+** and **CMD-** is clamped to almost zero, and the shutdown outputs are activated on all axes. *Clamping occurs independent of the microprocessor and the DSP.* (The clamping circuit is also connected to the watchdog timer; if the controller's microprocessor fails, the analog output voltage will be clamped.)	Product's *Installation Guide*
Shutdown Outputs	The controller uses the shutdown outputs to disable the attached drive if it detects a problem. *Shutdown outputs are not available on the OEM-AT6400 product.*	Product's *Installation Guide*
Drive Fault Inputs	The drive fault (**DFT**) inputs, found on the product's screw-terminal connectors, allows the drives to tell the controller if they encounter a fault condition. When a drive fault occurs, the controller stops motion (at the rate set with the LHAD command) and terminates program execution. No drive shutdown will result unless it is initiated with an ERRORP error program. *Drive fault inputs are not available on the OEM-AT6400 product.*	Product's *Installation Guide*
End-of-travel Limit Inputs	End-of-travel limits prevent the load from crashing through mechanical stops, an incident that can damage equipment and injure personnel. Use hardware or software limits, as your application requires.	*End-of-Travel Limits* (page 90)
User Fault Input	Using the INFNCi-F command, you can assign any of the programmable inputs the *user fault* function. You can then wire the input to activate when an external event, considered a *fault* by the user, occurs.	*Input Functions* (page 112)
Maximum Allowable Position Error (**servos only**)	A *position error* (TPER) is defined as the difference between the commanded position (TPC) and the actual position as measured by the feedback device (TFB). The maximum allowable position error is set with the SMPER command. When the maximum allowable position error is exceeded (usually due to instability or loss of position feedback), the controller shuts down the drive and sets error status bit #12 (reported by the TER command). If SMPER is set to zero (SMPERØ), position error will not be monitored.	*Servo Setup* (page 99)

☞ *Programmed Error-Handling Responses*

When any of the safety features listed above are exercised (e.g., **DFT** input is activated, etc.), the controller considers it an error condition. With the exception of the shutdown output activation, you can enable the ERROR command to check for the error condition, and when it occurs to branch to an assigned ERRORP program. Refer to *Error Handling* (page 30) for further information.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Options Overview (application examples)

The following are some application examples for the basic user interface options. Your application may require any one or combination of these options.

Stand-Alone Interface Options

After defining and storing controller programs, the controller can operate in a stand-alone fashion. A program stored in the controller may interactively prompt the user for input as part of the program (input via I/O switches, thumbwheels, RP240, joystick). A joystick can be use for situations requiring manual manipulation of the load.

Option	Application Example
Programmable I/O Switches, Thumbwheels (see page 128)	**Cut-to-length**: Load the stock into the machine, enter the length of the cut on the thumbwheels, and activate a programmable input switch to initiate the predefined cutting process (axis #1). When the stock is cut, a sensor activates a programmable input to stop the cutting process and the controller then initiates a predefined program that indexes the stock forward (axis #2) into position for the next cut.
RP240 *for stand-alone products* (see page 130)	**Grinding**: Program the RP240 function keys to select certain part types, and program one function key as a GO button. Select the part you want to grind, then put the part in the grinding machine and press the GO function key. The controller will then move the machine according to the predefined program assigned to the function key selected.
Joystick (see page 138)	**X-Y scanning/calibration**: Enter the joystick mode and use the 2-axis joystick to position an X-Y table under a microscope to arbitrarily scan different parts of the work piece (e.g., semi-conductor wafer). You can record certain locations to be used later in a motion process (e.g., for drilling, cutting, or photographing the work piece). The *Variable Arrays* section on page 120 provides an example using the joystick to teach positions. (Joystick interface is not available for the OEM-AT6400.)
ANI Analog Inputs *for servos with ANI option* (see page 142)	**Injection Molding**: Use for feedback from a pressure sensor to maintain constant, programmable force.

Programmable Logic Controller

The controller's programmable I/O may be connected to most PLCs. After defining and storing controller programs, the PLC typically executes programs, loads data, and manipulates inputs to the controller. The PLC instructs the controller to perform the motion segment of a total machine process.

EXAMPLE (X-Y point-to-point): A PLC controls several tools to stack and bore several steel plates at once. The controller is programmed to move an X-Y table in a pre-programmed sequence. The controller moves the load when the inputs are properly configured, signals the PLC when the load is in position, and waits for the signal to continue to the next position.

Host Computer Interface

A computer may be used to control a motion or machine process. A PC can monitor processes and orchestrate motion by sending motion commands to the controller or by executing motion programs already stored in the controller. This control might come from a BASIC or C program. A BASIC program example is provided on page 143.

Custom Graphical User Interfaces (GUIs)

Compumotor provides several tools you can use to create your own custom graphical user interfaces (GUIs). More detailed descriptions are provided on page 144.

- DLL (dynamic link library): Provided in ship kit. See page 51 for instructions.
- Motion OCX Toolkit™: OCX controls for Windows 95 and Windows NT.
- DDE6000™: DDE server.
- Motion Toolbox™: Library of LabVIEW® virtual instruments (VIs).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Programmable I/O Functions

Programmable inputs and outputs are provided to allow the controller to detect and respond to the state of switches, thumbwheels, electronic sensors, and outputs of other equipment such as drives and PLCs. Listed below are the programmable functions that may be assigned to the programmable I/O.

Programmable I/O offering differs by product. The total number of 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, the AT6400's TRG-A trigger input is represented by programmable input bit #25, but the ZETA6104's TRG-A trigger input is programmable input 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 the table on page 107.

NOTE

Refer to page 106 for instructions on establishing programmable input and output functions. Instructions for connecting to I/O devices are provided in your product's *Installation Guide*.

Input Functions

Related Command *	Function *(applicable to all products)*
`INFNCi-A`	No Function (**default**)
`INFNCi-B`	BCD Program Select
`INFNCi-C`	Kill
`INFNCi-D`	Stop
`INFNCi-E`	Pause/Continue
`INFNCi-F`	User Fault
`INFNCi-G`	<reserved>
`INFNCi-H`	Trigger Interrupt for position capture, registration, or `TRGFN` functions (trigger inputs only)
`INFNCi-I`	Interrupt to PC-AT (bus-based controllers only)
`INFNCi-J`	Jog+ (positive direction)
`INFNCi-K`	Jog- (negative direction)
`INFNCi-L`	Jog Speed Select
`INFNCi-P`	Program Select
`INFNCi-Q`	Program Security

Output Functions

Related Command *	Function	Servo Products	Stepper Products
`OUTFNCi-A`	Programmable Output (**default function**)	✓	✓
`OUTFNCi-B`	Moving/Not Moving (or In Position)	✓	✓
`OUTFNCi-C`	Program in Progress	✓	✓
`OUTFNCi-D`	End-of-Travel Limit Encountered	✓	✓
`OUTFNCi-E`	Stall Indicator	n/a	✓
`OUTFNCi-F`	Fault Indicator (indicates drive fault input or user fault input is active)	✓	✓
`OUTFNCi-G`	Position Error Exceeds Max. Limit	✓	n/a
`OUTFNCi-H`	Output on Position (auxiliary outputs only)	✓	n/a

* The "`i`" in the command syntax represents the number of the programmable input (e.g., `INFNC8-F` assigns general-purpose input #8 the "user fault" function). Bear in mind that the numbering scheme for programmable inputs and outputs varies by product (see page 107).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Thumbwheels

You can connect the controller's programmable I/O to a bank of thumbwheel switches to allow operator selection of motion or machine control parameters.

The controller allows two methods for thumbwheel use. One method uses Compumotor's TM8 thumbwheel module or IM32 input module. The other allows you to wire your own thumbwheels.

The TM8 requires a multiplexed BCD input scheme to read thumbwheel data. Therefore, a decode circuit must be used for thumbwheels. Compumotor recommends that you purchase Compumotor's TM8 module if you desire to use a thumbwheel interface. The TM8 contains the decode logic; therefore, only wiring is needed.

The commands that allow for thumbwheel data entry are:

`INSTW`.....Establish thumbwheel data inputs (TM8)
`OUTTW`.....Establish thumbwheel data outputs (TM8)
`TW`.....Read thumbwheels or PLC inputs
`INPLC`.....Establish PLC data inputs (Other thumbwheel module)
`OUTPLC` .....Establish PLC data outputs (Other thumbwheel module)

Using the TM8 & IM32 Modules

To use Compumotor's TM8 or IM32 Modules, follow the procedures below.

Step 1 Wire your TM8 or IM32 module to the controller as shown in your product's ***Installation Guide***.

Step 2 Use the commands below to configure your controller:

```
OUTTW1,1-3,0,10   ; Configure thumbwheel output set 1: outputs 1-3 are
                  ; strobe outputs, 10 ms strobe time per digit read.
                  ; Strobe time is 10 ms (minimum recommended for TM8).
INSTW1,1-4,5      ; Configure thumbwheel input set 1:
                  ; inputs 1-4 are data inputs, input 5 is a sign input.
INLVL00000        ; Inputs 1-5 configured active low
```

Step 3 Set the thumbwheel digits on your TM8 module to **+12345678**. To verify that you have wired your TM8 module(s) correctly and configured your controller I/O properly, enter the following commands:

```
VAR1=TW1          ; Assign data from all 8 thumbwheel digits to VAR1
VAR1              ; Displays the variable (*VAR1=+0.12345678). If you do not
                  ; receive this response, return to step 1 and retry.
```

Using your own Thumbwheel Module

As an alternative to Compumotor's TM8 Module, you can use your own thumbwheels. The controller's programming language allows direct input of BCD thumbwheel data via the programmable inputs. Use the steps below to set up and read the thumbwheel interface. Refer to the ***6000 Series Software Reference*** for descriptions of the commands used below.

Step 1 Wire your thumbwheels according to the schematic diagram provided in your product's ***Installation Guide***.

Step 2 Set up the inputs and outputs for operation with thumbwheels. The data valid input will be an input which the operator holds active to let the controller read the thumbwheels. This input is not necessary; however, it is often used when interfacing with PLCs.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
OUTPLC1,1-4,0,12  ; Config PLC output set 1: outputs 1-4 are strobe outputs,
                  ; no output enable bit, 12 ms strobe time per digit read.
INPLC1,1-8,9      ; Configure PLC input set 1: inputs 1-8 are data inputs,
                  ; input 9 is a sign input, no data valid input.
INLVL000000000    ; Inputs 1-9 configured active low
```

Step 3 The thumbwheels are read sequentially by outputs 1-4, which strobe two digits at a time. The sign bit is optional. Set the thumbwheels to **+12345678** and type in the following commands:

```
VAR1=TW5          ; Assign data from all 8 thumbwheel digits to VAR1
VAR1              ; Displays the variable (*VAR1=+0.12345678). If you do not
                  ; receive this response, return to step 1 and retry.
```

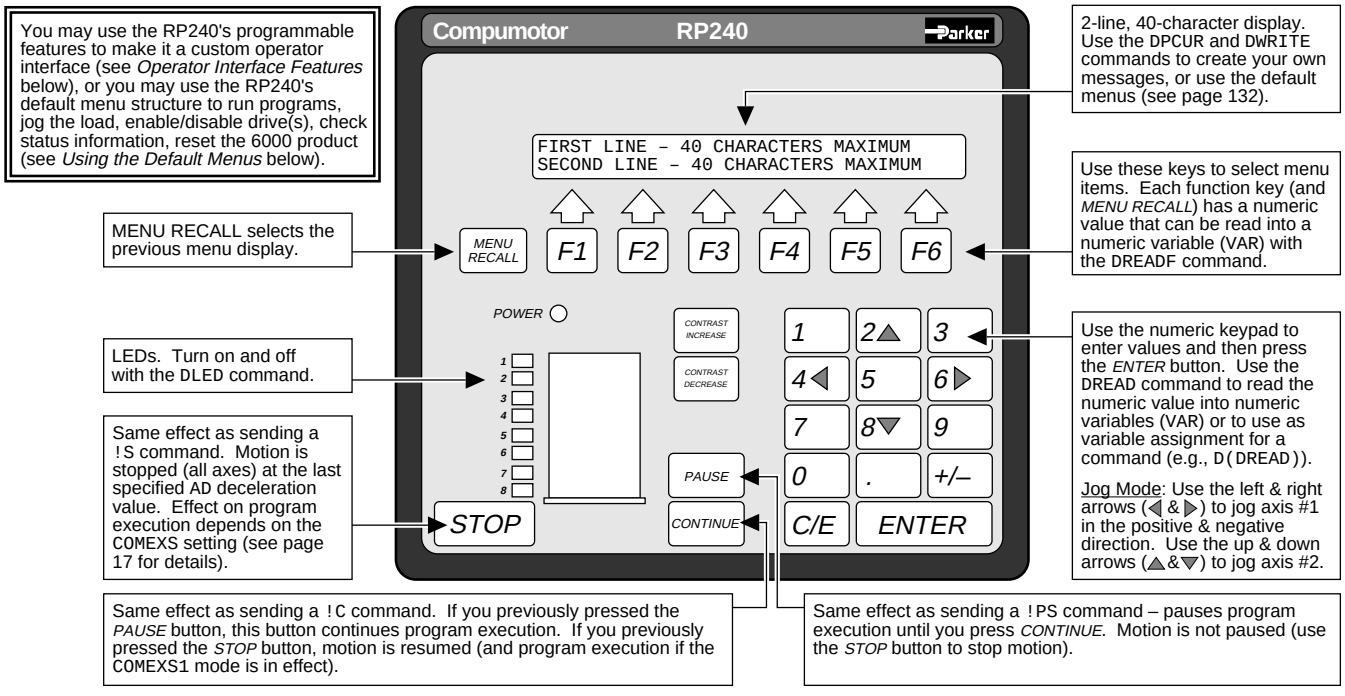
PLCs

The controller's programmable I/O may be connected to most PLCs. After defining and storing controller programs, the PLC typically executes programs, loads data, and manipulates inputs to the controller. The PLC instructs the controller to perform the motion segment of a total machine process.

Refer to your product's Installation Guide for instructions on connecting to I/O devices. For higher current or voltages above 24VDC, use external signal conditioning such as OPTO-22 compatible I/O signal conditioning racks. Contact your local distributor or automation technology center for information on these products.

RP240 Remote Operator Panel

6000 Series stand-alone products are directly compatible with the Compumotor RP240 Remote Operator Panel. This section describes how to use your 6000 product with the RP240. Instructions for connecting the RP240 are provided in your product's *Installation Guide*. Refer to the *Model RP240 User Guide* (p/n 88-012156-01) for information on RP240 hardware specifications, mounting guidelines, environmental considerations, and troubleshooting.



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Configuration

NOTE

As shipped from the factory, you can operate the RP240 from your 6000 product's "COM 2" or "RP240" port. This should be appropriate for the majority of applications.

For more information on controlling your product's multiple serial ports, refer to page 70.

Every stand-alone 6000 Series product has two serial ports. On existing 6000 products, the RS-232 connector (or Rx, Tx, and GND terminals on an AUX connector) is referenced as the "COM1" serial port, and the RP240 connector is referenced as the "COM2" serial port. Newer products have connectors labeled "COM1" (factory default function is RS-232) and "COM2" (factory default function is RP240).

To configure the 6000 product's serial ports for use with the RP240 and/or 6000 language commands, use the DRPCHK command. Be sure to select the affected serial port (**COM 1** or **COM 2**) with the PORT command before you execute the DRPCHK command. Once you issue the DRPCHK command, it is automatically saved in non-volatile memory. The configuration options are:

DRPCHKØ.....Use the serial port for 6000 commands only (default for **COM 1**)

DRPCHK1.....Check for RP240 on power up or reset. If detected, initialize RP240. If no RP240, use serial port for 6000 commands.

DRPCHK2.....Check for RP240 every 5-6 seconds. If detected, initialize RP240. Do not use port for 6000 commands.

DRPCHK3.....Check for RP240 on power up or reset. If detected, initialize RP240. If no RP240, use serial port for DWRITE command only. The DWRITE command can be used to transmit text strings to remote RS-232C devices. (default setting for **COM 2**)

Example **COM 2** is to be used for RS-485; **COM 1** is to be used for RP240, but the RP240 will be plugged in on an as-needed basis. The set-up commands for such an application should be executed in the following order:

```
PORT1       ; Select COM1 serial port for setup
DRPCHK2     ; Configure COM1 for RP240, periodic check
PORT2       ; Select COM2 serial port for setup
DRPCHKØ     ; Configure COM2 for 6000 commands only
```

Operator Interface Features

The RP240 may be used as your product's *operator interface*, not a program entry terminal. As an operator interface, the RP240 offers the following features:

- Displays text and variables
- 8 LEDs can be used as programmable status lights
- Operator data entry of variables: read data from RP240 into variables and command value substitutions (see substitutions table in Appendix C of ***Software Reference***)

Typically the user creates a program in the 6000 controller to control the RP240 display and RP240 LEDs. The program can read data and make variable assignments via the RP240's keypad and function keys.

The 6000 Series software commands for the RP240 are listed below. Detailed descriptions are provided in the ***6000 Series Software Reference***. The example below demonstrates the majority of the 6000 Series commands for the RP240.

DCLEAR.....Clear The RP240 Display
DJOG.....Enter RP240 Jog Mode
DLED.....Turn RP240 LEDs On/Off
DPASS .....Change RP240 Password
DPCUR .....Position The Cursor On The RP240 Display
[DREAD] .....Read RP240 Data
[DREADF] .....Read RP240 Function Key
DREADI.....RP240 Data Read Immediate Mode
DRPCHK.....Check for RP240
DVAR.....Display Variable On RP240
DWRITE.....Display Text On The RP240 Display

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Programming Example

```
DEF panel1                          ; Define program panel1
REPEAT                              ; Start of repeat loop
 DCLEAR0                            ; Clear display
 DWRITE"SELECT A FUNCTION KEY"      ; Display text "SELECT A FUNCTION KEY"
 DPCUR2,2                           ; Move cursor to line 2 column 2
 DWRITE"DIST"                       ; Display text "DIST"
 DPCUR2,9                           ; Move cursor to line 2 column 9
 DWRITE"GO"                         ; Display text "GO"
 DPCUR2,35                          ; Move cursor to line 2 column 35
 DWRITE"EXIT"                       ; Display text "EXIT"
 VAR1 = DREADF                      ; Input a function key
 IF (VAR1=1)                        ; If function key #1 hit
   GOSUB panel2                     ; GOSUB program panel2
  ELSE                              ; Else
  IF (VAR1=2)                       ; If function key #2 hit
   DLED1                            ; Turn on LED #1
   GO1                              ; Start motion on axis #1
   DLED0                            ; Turn off LED #1
  NIF                               ; End of IF (VAR1=2)
 NIF                                ; End of IF (VAR1=1)
UNTIL (VAR1=6)                      ; Repeat until VAR1=6 (function key 6)
DCLEAR0                             ; Clear display
DWRITE"LAST FUNCTION KEY = F"       ; Display text "LAST FUNCTION KEY = F"
DVAR1,1,0,0                         ; Display variable 1
END                                 ; End of panel1

DEF panel2                          ; Define prog panel2
DCLEAR0                             ; Clear display
DWRITE"ENTER DISTANCE"              ; Display text "ENTER DISTANCE"
D(DREAD)                            ; Enter distance number from RP240
END                                 ; End of panel2
```

Using the Default Menus

On power-up, the 6000 product will automatically default to a mode in which it controls the RP240 with the menu-driven functions listed below.

The flow chart below illustrates the RP240's menu structure in the default operating mode (when no 6000 product user program is controlling the RP240). Press the ***Menu Recall*** key to back up to the previous screen. The menu functions are described in detail below.

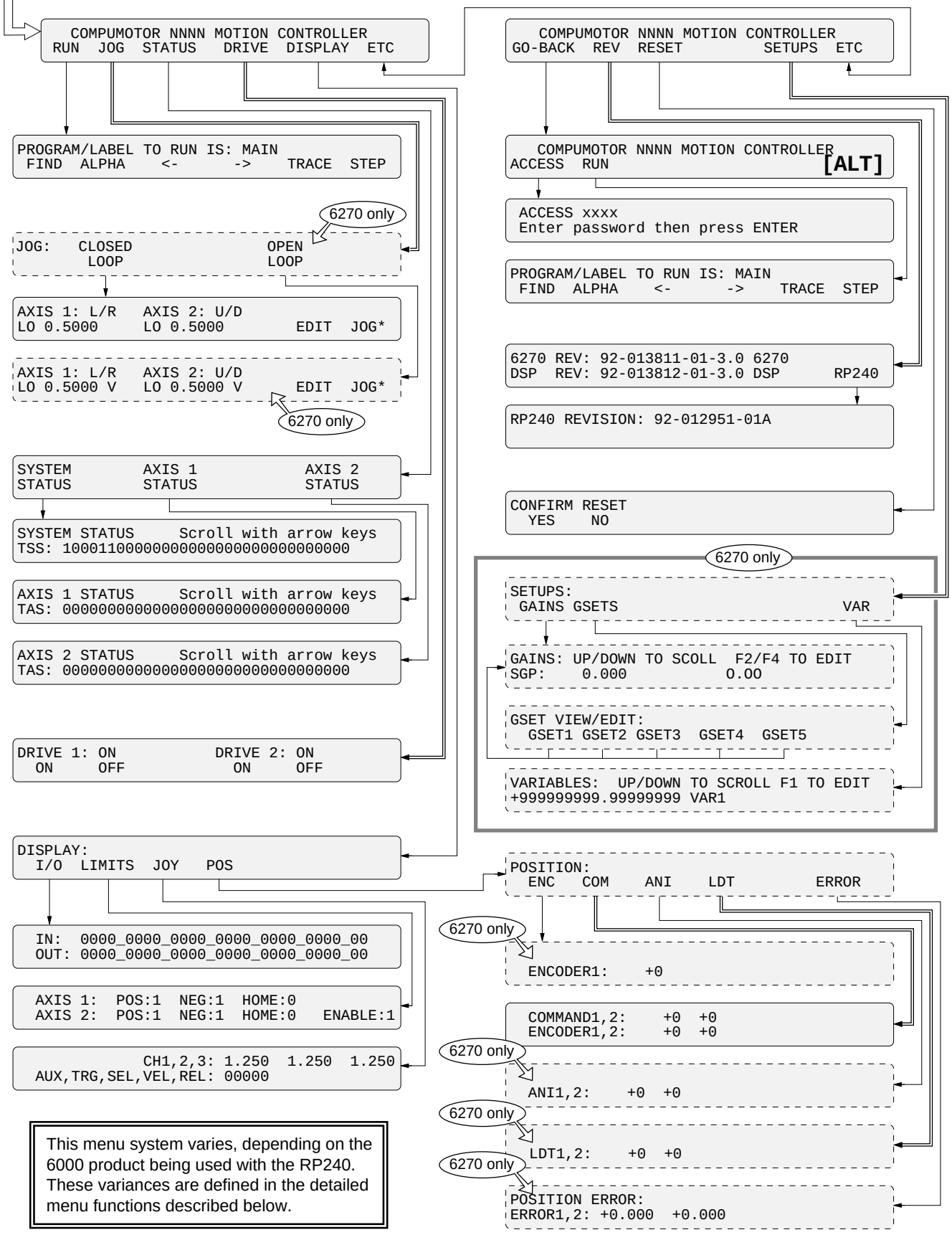
- Run a stored program (RUN, STOP, PAUSE and CONTINUE functions)
- Jog the load
- Display the status of:
  - System (TSS)
  - Axis (TAS)
  - Extended Axis (TASX) – 6104 only
  - I/O (TIN and TOUT)
  - Limits (TLIM) and P-CUT or ENBL input (TINO bit #6)
  - Position: Motor *for steppers* (TPM), Commanded *for servos* (TPC), Encoder (TPE) 6270 only: Current feedback device (TFB), ANI volts (TPANI), LDT (TLDT)
  - Firmware revision levels for the 6000 product (TREV) and the RP240
- Enable or disable the internal drive (DRIVE)
- Access RP240 menu functions with a security password (set with DPASS)
- RESET the 6000 product
- 6270 only: View and edit tuning gains (SGP, SGI, etc.), gain sets (SGSET), and numeric variables (VAR) and view only string variables (VARS)

NOTE: To disable these menus, the start-up program (the program assigned with the STARTP command) must contain the DCLEARØ command.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Default Power-up Menu

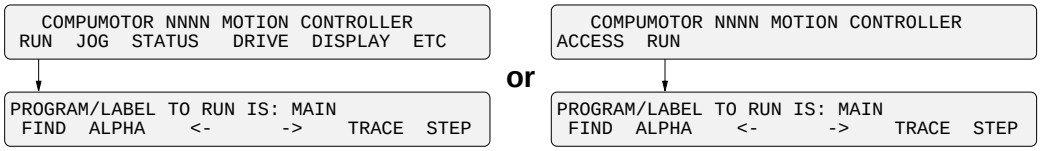
- If you change the password with the DPASS command, the menu marked with **[ALT]** becomes the power-up menu. The default password is the root name for your product (e.g., "6250" for the OEM6250, "6104" for the ZETA6104).
- Arrows indicate the menu path when you press the corresponding function key below the menu item.
- To back up to the previous menu item, press the *MENU RECALL* button.



This menu system varies, depending on the 6000 product being used with the RP240. These variances are defined in the detailed menu functions described below.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Running a Stored Program



After accessing the RUN menu, press ***F1*** to "find" the names of the programs stored in the 6000 product's memory; pressing ***F1*** repeatedly displays subsequent programs in the order in which they were stored in BBRAM. To execute the program, press the ***ENTER*** key.

To type in a program name at the location of the cursor, first select alpha or numeric characters with the ***F2*** function key (characters will be alpha if an asterisk appears to the right of ALPHA, or numeric if no asterisk appears). If alpha, press the up (2) or down (8) keys to move through the alphabet, if numeric, press the desired number key. Press ***F3*** to move the cursor to the left, or ***F4*** to move the cursor to the right.

Only user programs defined with DEF and END may be executed from this menu. Compiled profiles (contouring and GOBUF profiles) cannot be executed from this men; they must be executed from the terminal emulator with the PRUN command, or you can place the PRUN (name of path) command in a user program and then execute that program from this menu.

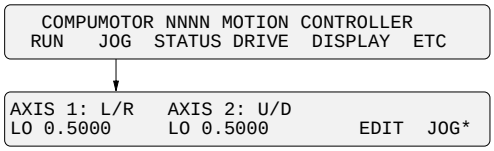
When a program is RUN and TRACE is selected (TRACE*), the RP240 display will trace all program commands as they are executed. This is different from the TRACE command in that the trace output goes to the RP240 display, not to a terminal via the serial port.

HINT: If you wish to display each command as it is executed, select STEP and TRACE and press the ENTER key to step through the program.

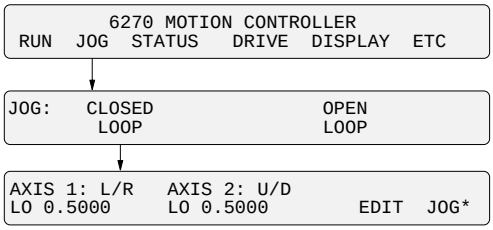
When a program is RUN and STEP is selected, step mode has been entered. This is similar to the STEP command, but when selected from the RUN menu the step mode allows single stepping by pressing the ***ENTER*** key. Both RP240 trace mode and step mode are exited when program execution is terminated.

Jogging

Most 6000 Products:



6270 Only:



You can jog individual axes by pressing the arrow keys on the RP240's numeric keypad. Pressing an arrow key on the numeric keypad will start motion and releasing the arrow key will stop motion. The left and right arrow keys correspond to axis #1 negative and positive direction, respectively. The up and down arrows keys are for axis #2 negative and positive direction, respectively.

The HI and LO values in the jog menu represent the velocity in units of revs/sec* (or inches/sec for LDT feedback or volts/sec for ANI feedback). If scaling is enabled, the value is multiplied by the programmed SCLV value.

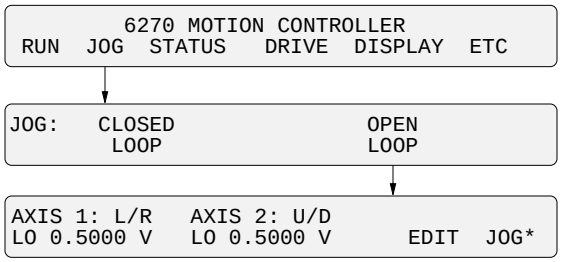
To edit the jog velocity* values:

1. Press the ***F5*** function key under EDIT (edit mode indicated with an asterisk).
2. Press the ***F1*** function key to select the HI and LO values (cursor appears under the first digit of the value selected).
3. Using the numeric keyboard, enter the value desired.
4. Repeat steps 2 and 3 for all values to be changed.
5. Press ***ENTER*** when finished editing.
6. To jog with the new velocity values, first press the ***F6*** function key (under JOG) to enable the arrow keys again.

Jog accel and decel values are specified by the JOGA and JOGAD commands, respectively.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Open Loop Jogging — 6270 ONLY



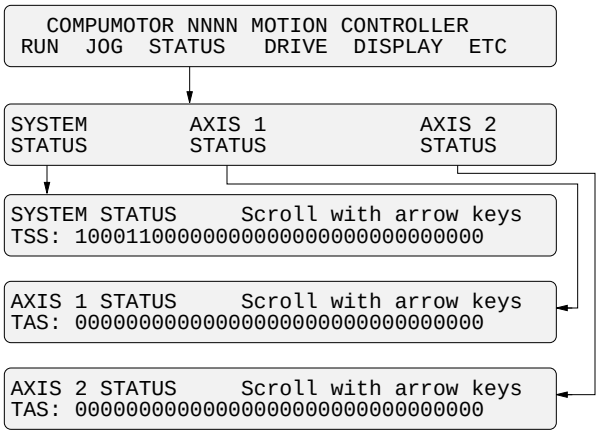
* The HI and LO values in the open-loop jog menu represent volts.

When using the open-loop jog mode, you should be aware of these conditions:

- **WARNING** — The hardware and software end-of-travel limits are disabled. Make sure it is safe to operate without end-of-travel limits before using the open-loop jog function.
- Gain values (SGILIM, SGAF, SGAFN, SGI, SGIN, SGP, etc.) set to zero (open-loop mode).
- 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 SFB0").
- SOFFS and SOFFSN set to zero, but allows subsequent servo offsets 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 mode, enabled with the KDRIVE command, so that the 6270 will shut down the valve/drive if a kill command (e.g., !K) is executed or if a kill input is activated. **CAUTION**: Shutting down a valve/cylinder system returns the valve to the null position; shutting down a rotary drive system allows the load to freewheel if there is no brake installed.

Status Reports: *System & Axis*

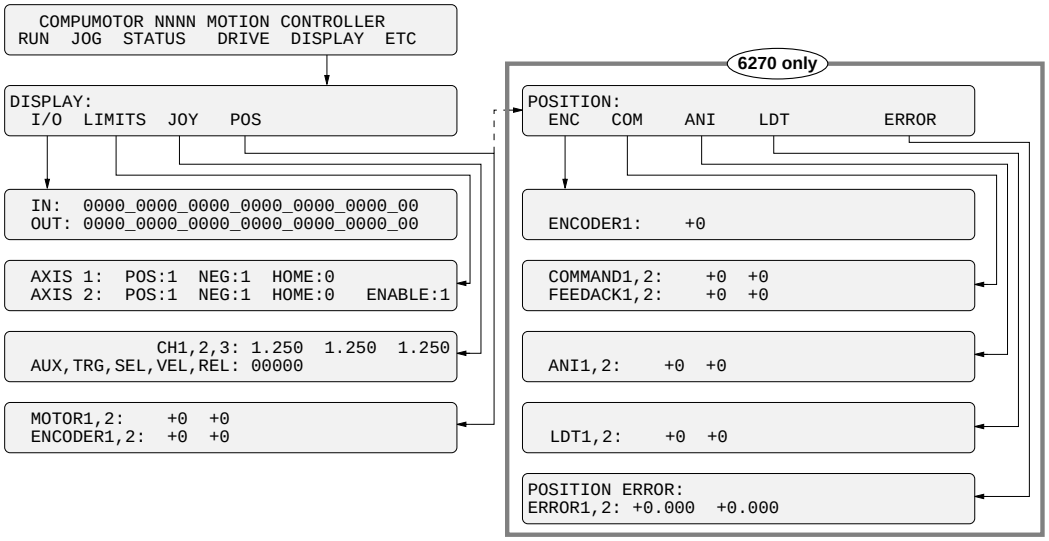


After accessing the desired status menu, you can ascertain the function and status of each system (TSS) or axis (TAS and TASX) status bit by pressing the arrow keys on the numeric keypad.

To view a more descriptive explanation of each status bit (includes a text description), press the left or right arrow keys on the numeric keypad.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Status Reports: *I/O, Limits, Position*



`I/O` Menu: Bit patterns for programmable inputs (including triggers) and programmable outputs (including auxiliary outputs) vary by product. Refer to page 107 to find the bit patterns for your product. As an example, the I/O bit pattern for the 6104 is as follows:

- Input bit pattern (left to right): Bits 1-16 are the general-purpose programmable inputs, bits 17 & 18 are triggers A and B (**TRG-A** and **TRG-B** on the **I/O** connector).
- Output bit pattern (left to right): Bits 1-8 are the general-purpose programmable outputs, bit 9 is auxiliary output A (**OUT-A** on the **I/O** connector).

`LIMITS` Menu:

- "`POS`" refers to the hardware end-of-travel limit imposed when counting in the positive direction. "`NEG`" refers to the limit imposed when counting in the negative direction.
- A "1" indicates that the input is grounded, "0" indicates not grounded.

  The end-of-travel limits (POS and NEG) must be grounded to allow motion (this is reversed if the active level is reversed with the `LHLVL` command).

  Steppers: The Pulse Cut input (**P-CUT** input terminal) must also be grounded before motion is allowed. When not grounded, step pulses are stopped independent of the 6000 product's microprocessor.
  Servos: The Enable input (**ENBL** input terminal) must also be grounded before motion is allowed. When not grounded, the analog output is held to zero volts and the shutdown outputs are activated.

`JOY` Menu:

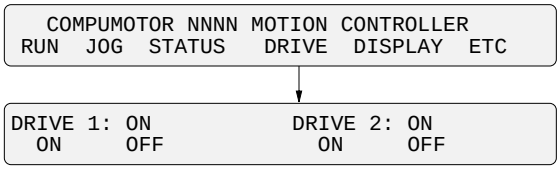
- The voltage levels present on the 3 analog channel on the 25-pin D **JOYSTICK** connector are listed.
- Also listed is the status of the joystick inputs ("1" indicates that the input is grounded/low, "0" indicates not grounded). `AUX`, `TRG`, `SEL`, `VEL`, and `REL` correspond respectively to pins 19, 18, 15, 16, and 17 on the **JOYSTICK** connector.

`POS` Menu:

- The position values shown are continually updated.
- Steppers: Reports the "`MOTOR`" position (always zero if in the encoder step mode – `ENC1`)
  Servos: Reports the "`COMMAND`" position.
- Position values (except for ANI) are subject to the `SCLD` scaling factor (if scaling is enabled—`SCALE1`), `PSET` offset value, feedback polarity (`ENCPOL`, `ANIPOL` and `LDTPOL`), and commanded direction polarity (`CMDDIR`). The ANI reading is always volts and is not scaled.
- 6270: "`POSITION ERROR`" = "`COMMAND`" position - actual ("`FEEDBACK`") position. "`FEEDBACK`" refers to the feedback device currently selected with the `SFB` command (default is LDT).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Enabling and Disabling the Drive(s)

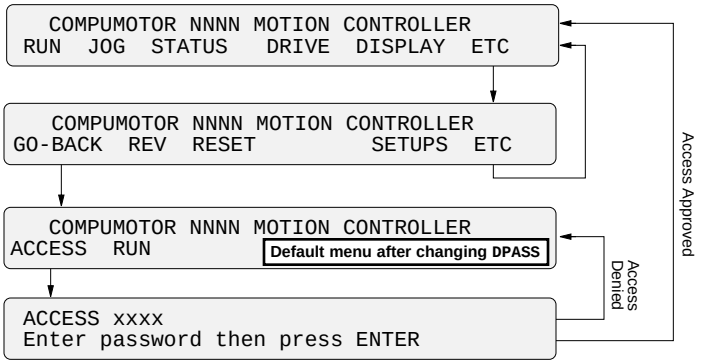


In the DRIVE menu, the current status of the drive(s) is displayed. To enable or disable the drive #1, press ***F1*** or ***F2***. To enable or disable the drive #2, press ***F3*** or ***F4***. This menu offers the same functionality as the DRIVE command.

WARNING

Shutting down a rotary drive system allows the load to freewheel if there is no brake installed. 6270: Shutting down a valve/cylinder system returns the valve to the null position.

Access Security

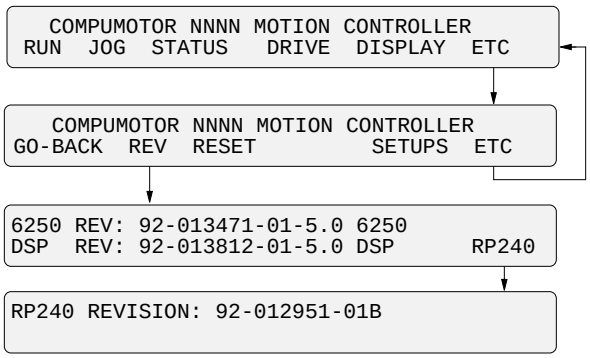


If the RP240 password is modified with the DPASS command to be other than the default (see *Changing the Password* below) the ACCESS menu then becomes the new default menu after power-up or executing a RESET.

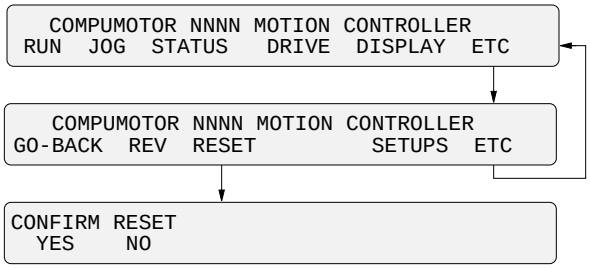
After that, the new password must then be entered to access the original default menu (see "Access Approved" path in illustration). If the operator does not know the new password, all he or she can do is run programs stored in the 6000 product (RUN).

Changing the Password: The default password is the root name for your product (e.g., "6250" for the OEM6250, "6104" for the ZETA6104, "6250" for the 6250). A new password (numeric value of up to 4 characters) can be established with the DPASS command. For example, the DPASS2001 command sets the password to 2001.

Revision Levels



Resetting the 6000 Product



After accessing the RESET menu, press the ***F1*** key to execute a reset (or press ***F2*** to cancel and exit the menu). The reset is identical to issuing a RESET command or cycling power to the 6000 product. If a start-up program has been assigned with the STARTP command, that program will be executed.

CAUTION

Executing a reset will restore most command values (exclusions: see page 33) to their factory default setting.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Joystick and Analog Inputs

☞ *Refer to your* **Installation Guide** *for connection procedures.*

The 6000 controller has up to four 8-bit analog input channels (CH1 - CH4). The analog inputs are configured as discrete single-ended inputs, with an input range of 0.0V to 2.5V. These inputs can be used to control an axis with a joystick (see *Joystick Control*). If you have a stepper product, you can use these inputs to scale velocity during feedrate override (see *Feedrate Override*). The voltage value on the analog inputs can be read using the ANV or TANV commands.

NOTE

The joystick and analog inputs feature is not available for the OEM-AT6400. 2-axis products have 3 analog input channels, 4-axis products have 4 channels.

Joystick Control

Joystick control can be achieved by simply connecting a joystick potentiometer to one of the analog inputs. Joystick operation is enabled with the JOY1 command.

NOTE: The Daedal JS6000 joystick is plug-compatible with the 6000 Series controllers. To order the JS6000, contact Daedal at (800) 245-6903 or contact your local distributor.

Travel limitations in potentiometers and voltage drops along the cables may make it impossible to achieve the full 0.0V to 2.5V range at the joystick input. Therefore, you must configure the controller to optimize the joystick's usable voltage range. This configuration will affect the *velocity resolution*. The velocity resolution is determined by the following equation:

$$\frac{\text{maximum velocity set with the JOYVH or the JOYVL command}}{\text{voltage range between the joystick's no-velocity region (}\textit{center deadband}\text{) and its maximum-velocity region (}\textit{end deadband}\text{)}}$$

To establish the velocity resolution, you must define the full-scale velocity and the usable voltage.

Define Full-Scale Velocity

You must define the full-scale velocity for your application with the JOYVH and JOYVL commands. Both commands establish the maximum velocity that can be obtained per axis by deflecting the potentiometer fully clockwise or fully counter-clockwise. The JOYVH command establishes the *high* velocity range (selected if the joystick select input is high – sinking current). The JOYVL command establishes the *low* velocity range (selected if the joystick select input is low – not sinking current).

The JOYAXL and JOYAXH commands define which analog channels are to be used with which joystick axes when the joystick select input is low or high, respectively.

Define Usable Voltage

Use the commands described in the table below to establish the joystick's usable velocity range.

The analog-to-digital converter is an 8-bit converter with a voltage range of 0.0V to 2.5V. With 8 bits to represent this range, there are 256 distinct voltage levels from 0.0V to 2.5V. 1 bit represents 2.5/256 or 0.00976 volts/bit.

Command	Name	Purpose
JOYEDB	End Deadband	This command defines voltage levels (shy of the 0.0V and 2.5V endpoints) at which maximum velocity occurs. Specifying an end deadband effectively decreases the voltage range of the analog input to compensate for joysticks that cannot reach the 0.0V and 2.5V endpoints.
JOYCTR (or JOYZ)	Center Voltage*	This command defines the voltage level for the center of the analog input range (the point at which zero velocity will result). As an alternative, you can use the JOYZ command, which reads the current voltage on the joystick input and considers it the center voltage. You can check the center voltage by typing in JOYCTR[cr].**
JOYCDB	Center Deadband	This command defines the voltage range on each side of the center voltage in which no motion will occur (allows for minor drift or variation in the joystick center position without causing motion).

* Because the center voltage can be set to a value other than the exact center of the potentiometer's voltage range, and because there could be two different velocity resolutions, the positive direction velocity resolution may be different than negative direction velocity resolution.

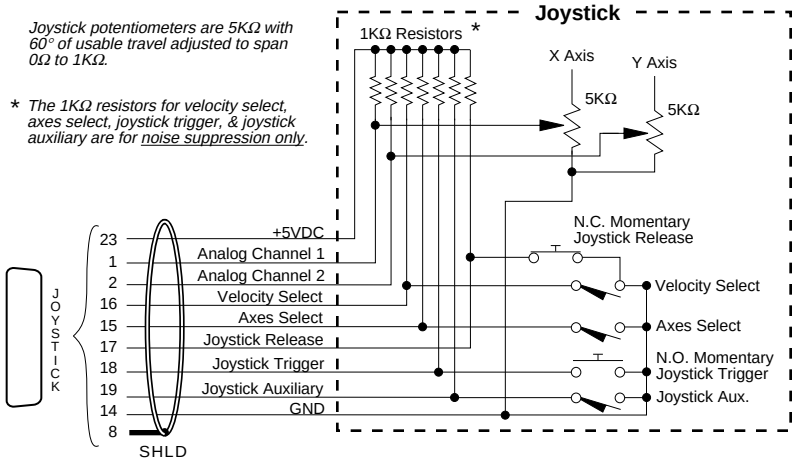
** Because of finite voltage increments, the controller may not report back exactly what you specified with the JOYCTR command.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Joystick Control Inputs

The table below describes the analog inputs available for joystick control (see diagram for connections). The status of each input is reported with the TINOF and TINO commands. The INO operator allows you to use an input's bit status in an assignment or comparison operation.

(see your product's Installation Guide *for input specifications)*



Joystick Input	Function
Axis Select	Using the JOYAXH and JOYAXL commands, you can establish two different configurations that specify the axes to be controlled by selected channels. The Axis Select input allows you to select the current configuration. Opening the Axis Select input (input is high, sinking current) selects the JOYAXH configuration. Closing the Axis Select input (input is low, not sinking current) selects the JOYAXL configuration. Example (4-axis controller moves 4 axes with one joystick): With axes select input high, analog channel #1 controls axis one and analog channel #2 controls axis two (JOYAXH1,2,Ø,Ø). With axes select input low, analog channel #1 controls axis three and analog channel #2 controls axis four (JOYAXLØ,Ø,1,2). Input State — Function — TINO/TINOF/INO Status Select JOYAXH configuration. — Bit #3 reports zero (Ø) Select JOYAXL configuration. — Bit #3 reports one (1)
Velocity Select	Using the JOYVH and JOYVL commands, you can establish a high-speed jog velocity and a low-speed jog velocity. The Velocity Select input allows you to select the current jog velocity. Opening the input (input is high, sinking current) selects the JOYVH configuration. Closing the input (input is low, not sinking current) selects the JOYVL configuration. The high range could be used to quickly move to a location, the low range could be used for accurate positioning. *When this input is not connected, the low velocity range is selected.* Input State — Function — TINO/TINOF/INO Status Select JOYVH configuration. — Bit #4 reports zero (Ø) Select JOYVL configuration. — Bit #4 reports one (1)
Release	The Joystick Release input allows you to indicate to the 6000 product that you have finished using the joystick and program execution may continue with the next statement. When a program enables joystick control of motion (with the JOY command), program execution will stop and then resume when the user is finished with joystick mode (assuming the Continuous Command Execution Mode is disabled with the COMEXCØ command). The joystick mode cannot be enabled while this input is open (high, sinking current). When you open the input, the joystick mode is disabled and can be re-enabled only with the JOY command. Input State — Function — TINO/TINOF/INO Status Disables joystick mode. — Bit #5 reports zero (Ø) Joystick mode can be enabled. — Bit #5 reports one (1)
Trigger (general-purpose)	The status of this input can be read by your 6000 program and may be used to control program flow (see INO command). Input State — Function — TINO/TINOF/INO Status Input is high, sinking current. — Bit #2 reports zero (Ø) Input is low, not sinking current. — Bit #2 reports one (1)
Auxiliary (general-purpose)	The status of this input can be read by your 6000 program and may be used to control program flow (see INO command). Input State — Function — TINO/TINOF/INO Status Input is high, sinking current. — Bit #1 reports zero (Ø) Input is low, not sinking current. — Bit #1 reports one (1)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Joystick Set Up Example

This example represents a typical two-axis joystick application in which a high velocity range is required to move to a region, then a low velocity range is required for a fine search. After the search is completed it is necessary to record the load positions, then move to the next region. The joystick trigger input can be used to indicate that the position should be read. The joystick release is used to exit the joystick mode and continue with the motion program.

The following table describes the requirements of the application (using rotary motors), and how the controller is configured to satisfy those requirements. The resulting joystick voltage configuration is illustrated below. One analog input channel is used for each axis.

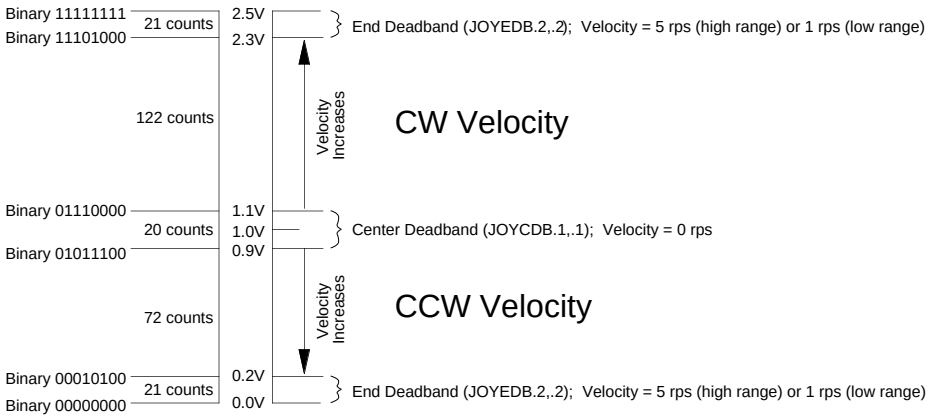
Requirement	Configuration
Set max. high-range velocity to 5 units/sec (on both axes)	Type in the `JOYVH5,5` command
Set max. low-range velocity to 1 units/sec (on both axes)	Type in the `JOYVL1,1` command
No velocity when voltage is at 1.0V	Set center voltage with `JOYCTR1,1,` command, or set voltage level at both analog inputs to 1.0V and type in `JOYZ11`
Joystick cannot reliably rest at 1.0V, but can rest within ±0.1V of 1.0V	Set center deadband of 0.1V with `JOYCDB.1,.1` command (0.1V is the system default)
Joystick can only produce maximum of 2.3V and minimum of 0.2V	Set end deadband to get max. velocity at 2.3V or 0.2V with the `JOYEDB.2,.2` command. Voltage range: CW = 1.1V to 2.3V (1.2V total) CCW = 0.9V to 0.2V (0.7V total) Voltage resolution: see below

The high-range velocity resolutions (at 5 rps max.) are calculated as follows:

$$\text{CW: } \frac{5 \text{ rps}}{\text{voltage range of 1.2V (122 counts)}} = 0.041 \text{ rps/count; CCW: } \frac{5 \text{ rps}}{\text{voltage range of 0.7V (72 counts)}} = 0.069 \text{ rps/count}$$

The low-range velocity resolutions (at 1 rps max.) are calculated as follows:

$$\text{CW: } \frac{1 \text{ rps}}{\text{voltage range of 1.2V (122 counts)}} = 0.008 \text{ rps/count; CCW: } \frac{1 \text{ rps}}{\text{voltage range of 0.7V (72 counts)}} = 0.014 \text{ rps/count}$$



Analog Voltage Override

Before you actually wire the analog inputs, you can simulate their activation in software by using the `ANVO` command. For instance, `ANVO1.2,1.6,1.8,1.3` overrides the hardware analog input channels—1.2V on channel 1, 1.6V on channel 2, 1.8V on channel 3, and 1.3V on channel 4. The `ANVO` values are used in any command or function that references the analog input channels, but only those channels for which `ANVOEN` is set to 1 (e.g., Given `ANVOENØ11Ø`, the `ANVO` values 1.6V and 1.8V are referenced for channels 2 and 3 only.).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Feedrate Override *(multi-axis steppers only)*

Feedrate override is used to synchronously scale all phases of motion on all axes (except distance). The amount of scaling is expressed in terms of percentage from 0 to 100. The percentage of feedrate can be controlled by an analog voltage or by the FRPER command.

When feedrate override is enabled, the frequency at which the controller's motion algorithm updates the velocity output varies according to the feedrate percentage specified. Without feedrate override, the motion algorithm is updated every 2 ms. At 100%, the velocity output is updated every 2 ms. At 50%, the velocity output is updated every 4 ms.

4-Axis Controllers

During feedrate override, axis 4 is used to perform the feedrate override function. This axis can no longer be used for motion, and must be disconnected.

Using Feedrate Override While Contouring

When you enter or exit the feedrate override mode with the FR command, you will have to recompile (PCOMP) any previously compiled contouring paths.

Hardware Feedrate Override (FR1)

To adjust feedrate using an analog voltage, enable feedrate control with the FR1 command. The lower the voltage, the lower the feedrate percentage. The higher the voltage, the higher the feedrate percentage. The end-deadband (JOYEDB) for the analog input channel is in effect during feedrate control, but the center-deadband (JOYCDB) has no effect. If the end-deadband is zero, ØVDC commands 0% feedrate, and 2.5VDC commands 100% feedrate. The velocity update (2 ms at 100%) can be calculated as follows: $\frac{2 * (2.5 - 2 * \text{JOYEDB})}{\text{Analog Voltage - JOYEDB}}$

```
FRA50000   ; Set the feedrate acceleration to 50000 percent/sec/sec
FRH1       ; Control axes with analog input 1 when Axes Select Input is closed
FRL2       ; Control axes with analog input 2 when Axes Select Input is open
FR1        ; Enable feedrate override
A100       ; Set axis 1 acceleration to 100 units/sec/sec
V10        ; Set axis 1 velocity to 10 units/sec
D100000    ; Set axis 1 distance to 100000 units
GO1        ; Initiate motion (axis 1). During motion, if the voltage on
           ; input 1 changes, the velocity of this move will change also.
```

Software Feedrate Override (FR2)

To adjust the feedrate using the FRPER command, enable feedrate control using the FR2 command. The feedrate percentage can then be specified directly. The command FRPER5Ø would set the feedrate to 50%, while FRPER1ØØ would set the feedrate to 100%. The velocity update (2 ms at 100%) can be calculated with the equation $\frac{2 * \text{FRPER}}{100}$.

The example below demonstrates using the software command FRPER to control axis 1 when feedrate override is enabled. Within the example, inputs 1, 2, and 3 are used to control the feedrate override percentage. When input #3 is activated, the original move velocity is doubled. When input #2 is activated, the move slows down to half its original velocity. Input #1 returns the move to its original velocity. Input #4 stops the move and continues command processing with the command after the NWHILE command.

```
DEF temp          ; Begin definition of program temp
FRA50000          ; Set the feedrate acceleration to 50000 percent/sec/sec
FRPER50           ; Feedrate override percentage equals 50%
FR2               ; Enable feedrate override
MC1               ; Enable continuous mode moves
A100              ; Set acceleration
V4                ; Set velocity
D+                ; Set direction to positive
COMEXC1           ; Enable continuous command processing mode
GO1               ; Initiate motion on axis 1
WHILE(IN=bXXX0)   ; Repeat commands (WHILE to NWHILE) until input #4 activates
 IF(IN=b1)        ; If input #1 is active, change feedrate percentage to 50%
   FRPER50        ; Set feedrate percentage to 50%
  NIF             ; End IF command
   IF(IN=bX1)     ; If input #2 is active, change feedrate percentage to 25%
    FRPER25       ; Set feedrate percentage to 25%
   NIF            ; End IF command
    IF(IN=bXX1)   ; If input #3 is active, change feedrate percentage to 100%
     FRPER100     ; Set feedrate percentage to 100%
 NIF              ; End IF command
NWHILE            ; End WHILE command
S1                ; Stop motion on axis 1
COMEXC0           ; Disable continuous command processing mode
END               ; End definition of program temp
RUN temp          ; Initiate program temp
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

ANI Analog Input Interface *(only servo controllers with ANI option)*

Refer to your product's **Installation Guide** *for ANI connection information.*

6000 Series servo controllers with the -ANI option offer ±10V, 14-bit analog inputs (referred to as "ANI" inputs). Each input has an anti-aliasing filter and is sampled at the servo sample rate (set with the SSFR command). The voltage value of the ANI inputs can be transferred to the terminal with the TANI command, or used in an assignment or comparison operation with the [ANI] command (e.g., IF(1ANI<2.4)). The position value of the ANI inputs can be transferred to the terminal with the TPANI command, or used in an assignment or comparison operation with the [PANI] command (e.g., WAIT(1PANI>421)).

Three common applications of the ANI inputs are:

- Position command to the control loop (e.g., as a master axis in Following mode)
- Position feedback to the control loop
- A force or torque feedback signal

Programming Example

The portion of 6000 code below (for two axes of control) demonstrates how to read the analog inputs into the controller and set the commanded analog output of each axis to that value. If you have a torque drive, this provides open-loop torque control.

```
SGP0,0              ; Turn off servo proportional feedback gain
SGI0,0              ; Turn off servo integral feedback gain
SGV0,0              ; Turn off servo velocity feedback gain
SGAF0,0             ; Turn off servo acceleration feedforward gain
SGVF0,0             ; Turn off servo velocity feedforward gain
SOFFS0,0            ; Set offset to zero (analog output will be 0 volts)
L                   ; Enter an infinite loop
 VAR1=1ANI          ; Read value of ANI analog input #1 into variable #1
 VAR2=2ANI          ; Read value of ANI analog input #2 into variable #2
 SOFFS(VAR1),(VAR2) ; Assign voltages from ANI analog inputs #1 & #2 to the
                    ; analog output for axes #1 & # 2, respectively
 T.01               ; Set time delay to 10 milliseconds
 LN                 ; End loop
```

4-20 mA Feedback

The analog inputs can be used to monitor a process using 4-20 mA feedback. This is accomplished by simply attaching a resistor from the **ANI** input to **AGND** (e.g., a 500Ω, 1% resistor would facilitate a 2-10V input). In this way, the current is converted to a voltage that can be monitored with the ANI input. The PSET command can be used to set the input readings to user coordinates.

ANI as a Feedback Device

The ANI analog inputs, when selected as a feedback source with the SFB command, is assumed to provide position information. With this feedback it is possible to solve applications that require positioning to a voltage, rather than positioning to a known position. Some example applications are as follows:

- Using a potentiometer as feedback (mechanical motion is mimicked by the 6000 controller)
- Maintaining a force while position changes due to fluid evacuating a chamber
- Opening or closing a valve as another process changes

Auxiliary Analog Output ("half axis" — AT6n50 only)

☞ *See* **Installation Guide** *for connection instructions.*

The AT6n50 offers an auxiliary analog output (**ANA**), located at terminal #3 on the **AUX** connector. This output provides ±10V with an accuracy of ±5%, and is derived from an 8-bit digital-to-analog converter.

To control the voltage at this output, use the OUTANA command. Syntax is OUTANA<r>, where <r> is the desired output in volts with a range of -10 to +10 (e.g., OUTANA5.ØØ sets the analog output to +5.00 volts). The default output is zero volts. The **ANA** output is not affected by the state of the enable (**ENBL**) input.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Another choice for a user interface is to use a host computer and execute a motion program using the RS-232C serial interface. A host computer may be used to run a motion program interactively from a BASIC or C program (high-level program controls the 6000 product and acts as a user interface). A BASIC program example (for the 6250 product) is provided below.

```
10  '      6000 Series Serial Communication BASIC Routine
12  '                        6000.BAS
14  '
16  ' ***********************************************************
18  '
20  ' This program will set the communications parameters for the
22  ' serial port on a PC to communicate with a 6000 series
24  ' stand-alone product.
26  '
28  ' ***********************************************************
30  '
100 '*** open com port 1 at 9600 baud, no parity, 8 data bits, 1 stop bit
110 '*** activate Request to Send (RS), suppress Clear to Send (CS), suppress
120 '*** DATA set ready (DS), and suppress Carrier Detect (CD) ***
130 OPEN "COM1:9600,N,8,1,RS,CS,DS,CD" FOR RANDOM AS #1
140 '
150 '*** initialize variables ***
160 MOVE$ = ""         ' *** commands to be sent to the product ***
170 RESPONSE$ = ""     ' *** response from the product ***
180 POSITION$ = ""     ' *** feedback position reported ***
190 SETUP$ = ""        ' *** setup commands ***
200 '
210 '*** format the screen and wait for the user to start the program ***
220 CLS : LOCATE 12, 20
230 PRINT "Press any key to start the program"
240 '
250 '*** wait for the user to press a key ***
260 PRESS$ = INKEY$
270 IF PRESS$ = "" THEN 260
280 CLS
290 '
300 '*** set a pre-defined move to make ***
310 SETUP$ = "ECHO1:ERRLVL0:LH0,0:"
320 MOVE$ = "A100,100:V2,2:D50000,50000:GO11:TFB:"
330 '
340 '
400 '*** send the commands to the product ***
410 PRINT #1, SETUP$
420 PRINT #1, MOVE$
430 '
500 '*** read the response from the TFB command ***
510 '   *** the controller will send a leading "+" or "-" in response to the TFB command to
520 '   *** indicate which direction travel has occurred. ***
530 WHILE (RESPONSE$ <> "+" AND RESPONSE$ <> "-") ' *** this loop waits for the "+"
540   RESPONSE$ = INPUT$(1, #1)                   ' *** or "-" characters to be returned
550 WEND                                          ' *** before reading the position ***
560 '
570 WHILE (RESPONSE$ <> CHR$(13))        ' *** this loop reads one character at a time
580   POSITION$ = POSITION$ + RESPONSE$  ' *** from the serial buffer until a carriage
590   RESPONSE$ = INPUT$(1, #1)          ' *** return is encountered ***
600 WEND
610 '
620 '*** print the response to the screen ***
630 LOCATE 12, 20: PRINT "Position is " + POSITION$
640 '
650 'END
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Graphical User Interface (GUI) Development Tools

TO ORDER

To order Motion OCX Toolkit™, DDE6000™, or Motion Toolbox®, contact your local Automation Technology Center (ATC) or distributor.

Dynamic Link Libraries (DLLs)

To help you develop your own Windows applications, Compumotor provides dynamic link libraries (DLLs) for Windows 3.1, Windows 95, and Windows NT. The DLLs contain communication functions for use with all of Compumotor's bus-based 6000 Series control products; functions include sending commands to the controller, fetching responses from the controller, and polling status information from the controller's fast status area.

For detailed information about the DLLs, see page 51.

- Windows 3.1 driver……WIN6400.DLL
- Windows 95 driver……WN956000.DLL
- Windows NT driver……NT6400.DLL

FREE: These DLLs are provided on your Motion Architect diskette. To install them, run the Motion Architect installer and select the desired DLLs from the "Custom Installation" dialog.

Motion OCX Toolkit™

The Motion OCX Toolkit provides 32-bit Ole Custom Controls (OCXs) designed to run under Windows 95 or Windows NT. Motion OCX Toolkit may be used with all of the 6000 Series bus-based products. Controls include:

- Communications Shell — control basic communication with the 6000 product, including interrupt handling and sending/receiving files.
- Fast-Status Polling — poll the 6000 product's fast status register (see page 43 for description of fast status area).
- Terminal — terminal emulator.

The OCX controls can be used with Visual Basic 4.0, Delphi 2.0. Visual C++ 4.x, or any 32 bit development environment that can contain OCX controls. With the Motion OCX Toolkit, you can quickly develop you own custom operator interface.

DDE6000™ Dynamic Data Exchange (DDE)

DDE6000 is a Dynamic Data Exchange (DDE) server that you can use to facilitate communication between a Windows application and your 6000 product. For example, you might use DDE6000 with a third-part factory automation software and operator interface, such as Wonderware's In-Touch™. DDE6000 supports NetDDE, which allows operation over a Windows for Workgroups, Windows 95, or Windows NT network.

☞ *Multiple 6000 products may be accessed simultaneously with the DDE6000.*

The DDE6000 *server*, a Windows program, provides access to 6000 controller data that can be useful to other Windows programs (DDE *clients*). DDE6000 supports three types of "conversations" with a DDE client:

- Cold Link …..Allows a client to directly request a particular data item from DDE6000.
- Hot Link.....Allows a client to be automatically updated when a particular data item from the DDE6000 has changed.
- Warm Link....Combination of cold link and hot link, where a client wants to be informed of changes in the DDE6000 data without immediately receiving the new data item.

Motion Toolbox™

Motion Toolbox is a library of LabVIEW® virtual instruments (VIs) for Compumotor's 6000 Series controllers. Motion Toolbox allows LabVIEW programmers to develop motion control systems for a wide range of applications, including automated test and manufacturing, medical and biotech, metering and dispensing, machine control, and laboratory automation. Motion Toolbox provides these capabilities:

- Motion control, including velocity, acceleration, deceleration, go, stop, kill, etc.
- Setup, control, and command file transfer
- Counter and timer configuration and control
- Indexer, encoder, and drive configuration
- Home, hardware limit, and soft limit configuration
- Jogging and joystick configuration
- I/O setup and function configuration
- Fast status querying of I/O, limit, home, motor and encoder position, velocity, etc.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

CHAPTER FIVE

5 Custom Profiling

IN THIS CHAPTER

This chapter explains how to use these custom profiling features:

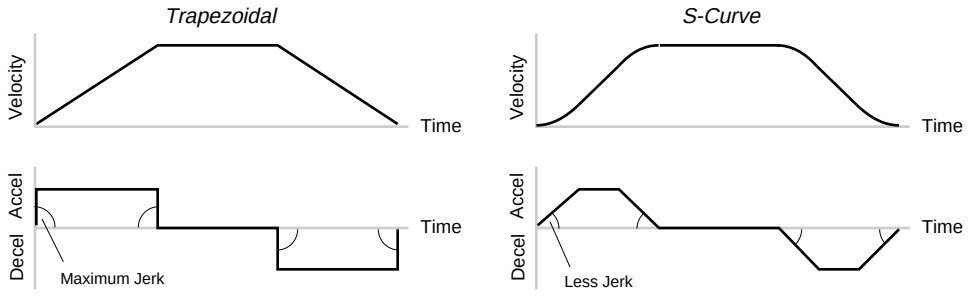
- S-Curve Profiling (servos only)..... 146
- Streaming Mode (bus-based steppers only)..... 148
- Linear Interpolation..... 152
- Contouring (circular interpolation)..... 153
- Compiled Motion Profiling..... 163
- On-the-Fly Motion (pre-emptive GOs)..... 178
- Registration..... 182
- Synchronizing Motion..... 186

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

S-Curve Profiling *(servos only)*

6000 servo controllers allow you to perform *S-curve* move profiles, in addition to the usual trapezoidal profiles. S-curve profiling provides smoother motion control by reducing the *jerk* (rate of change) in acceleration and deceleration portions of the move profile (see drawing below).

S-curve profiling is not available during Contouring.



S-curves improve position tracking.

Because S-curve profiling reduces jerk, it improves position tracking performance in servo systems, especially in linear interpolation applications (not contouring).

S-curve programming requirements.

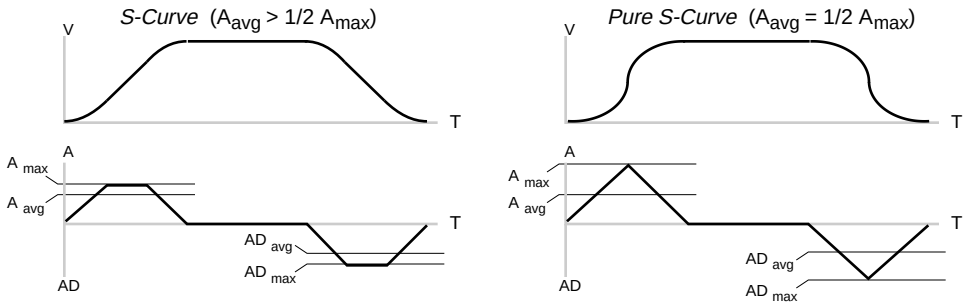
To program an S-curve profile, you must use the *average accel/decel* commands provided in the 6000 Series programming language. For every maximum accel/decel command (e.g., A, AD, HOMA, HOMAD, JOGA, JOGAD, etc.) there is an *average* command for S-curve profiling (see table below).

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

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

- If $A_{avg} > 1/2\ A_{max}$, but $A_{avg} < A_{max}$, you have achieved an S-curve profile with a variable period of constant accel/decel (see drawing below).
- If $A_{avg} = 1/2\ A_{max}$, you have achieved what is called a *Pure S-curve* profile in which there is no period of constant accel/decel and jerk is at an absolute minimum (see drawing below).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



- Once you enter an A_{avg} value that is ≠ zero and satisfies $1/2\ A_{max} \leq A_{avg} < A_{max}$, S-curve profiling is enabled, but only in the operation that uses that particular A_{avg} command. For example, entering a `HOMAA` command enables S-curve acceleration profiling only for homing moves, not for other functions such as jogging (which would require the `JOGAA` command). To return to the default trapezoidal profiling mode, enter an A_{avg} value of zero, or set $A_{avg} = A_{max}$.

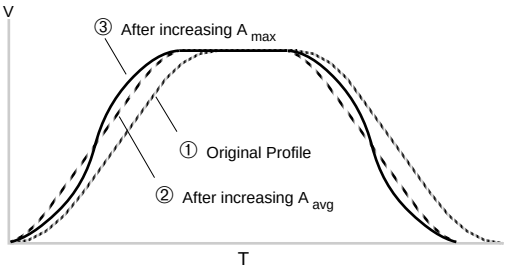
- If $A_{avg} = A_{max}$, a trapezoidal profile results, but can be changed to an S-curve by specifying a new A_{avg} value less than A_{max}, or set A_{max} greater than A_{avg}.

- If $A_{avg} < 1/2\ A_{max}$, or $A_{avg} > A_{max}$, when you try to initiate motion, the move will not be executed and an error message, `*INVALID CONDITIONS FOR S_CURVE ACCELERATION–FIELD` *n*, will be displayed.

- If A_{avg} = zero or if you never enter an A_{avg} command, the controller defaults to trapezoidal profiling and the A_{avg} command value will always match the A_{max} command value. However, if you enter an A_{avg} **deceleration** of zero, you will receive the error message `*INVALID DATA–FIELD n`, where n is the number of the data field.

- If you never enter the maximum (A_{max}) or average (A_{avg}) decel command values (`AD` or `ADA`, `HOMAD` or `HOMADA`, etc.), the average decel value will always match, or *track*, the average accel value (`AA`, `HOMAA`, etc.). However, once you change the maximum decel, the average decel will no longer track the average accel.

- If you increase the A_{avg} value above the pure S-curve level ($A_{avg} > 1/2\ A_{max}$), the time required to reach the target velocity and the target distance decreases; however, increasing A_{avg} also increases jerk. After increasing A_{avg}, you can reduce the jerk by increasing A_{max} (see illustration); however, increasing A_{max} requires a greater torque to achieve the commanded velocity at the mid-point of the acceleration profile.



- You can calculate the profile's accel/decel time with the following equations (calculation method is identical for S-curve **and** trapezoidal):

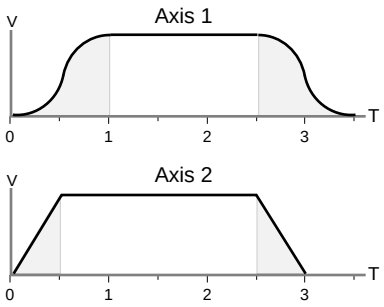
$$\text{Time of accel or decel} = \frac{\text{Velocity}}{A_{avg}} \quad \text{or} \quad \text{Time of accel or decel} = \sqrt{\frac{2 * \text{Distance}}{A_{avg}}}$$

- Accel/decel scaling (`SCLA` or `PSCLA`) affects A_{avg} the same as it does for A_{max}, regardless of if the profile is trapezoidal or S-curve (see *Scaling* section in Chapter 3).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Example In this example, Axis 1 executes a pure S-curve profile that takes 1 second to reach a velocity of 5 rps and 1 second to return to zero velocity. Axis 2 executes a trapezoidal profile that takes 0.5 seconds to reach a velocity of 5 rps and 0.5 seconds to return to zero velocity.

```
@SFB1       ; Select encoder feedback
@ERES4000   ; Set resolution to 4000
            ; steps/rev
SCALE0      ; Disable scaling
@MA0        ; Select incremental
            ; positioning mode
@D50000     ; Set distances to 50,000
            ; positive counts
A10,10      ; Set max. accel to 10
            ; revs/sec/sec (both axes)
AA5,10      ; Set avg. accel to 5
            ; revs/sec/sec on axis 1, and
            ; 10 revs/sec/sec on axis 2
AD10,10     ; Set max. decel to 10 revs/sec/sec (both axes)
ADA5,10     ; Set avg. decel to 5 r/s/s on axis 1, and 10 r/s/s on axis 2
V5,5        ; Set velocity to 5 revs/sec on both axes
GO11        ; Execute motion on both axes
```



Timed Data Streaming *(bus-based steppers only)*

The Timed Data Streaming (*Streaming*) modes allow you precise multi-axis distance and velocity control. Data streaming is accomplished by dividing the motion profile into small straight-line segments, allowing you to control the profile shape with greater accuracy.

Time-distance streaming (STREAM1) allows you to control the number of steps output over a given time period.

Time-velocity streaming (STREAM2) allows you to control the frequency of the step output over a given period of time.

To produce a data streaming profile, you must do the following:

1. Establish the streaming update interval (STD). This interval can be any multiple of 2 beginning with 10 milliseconds and ending with 50.
2. Enable the desired streaming mode. STREAM1 for time-distance streaming, or STREAM2 for time-velocity streaming.
3. Send datapoints via SD commands. Streaming Data (SD) commands allow you to change distance or velocity values and enable certain streaming functions (see table below). The SD command syntax is SD<i>,<i>,<i>,<i>, where each data or function assignment (<i>) represents one *datapoint*. As many as four datapoints are possible per SD command—one for each axis. The nine types of datapoints are listed in the table below.

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-travel 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 4000000012 must be entered in the third axis' data field (SD,,4000000012).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Greater in-depth discussions on each SD command function are provided in the ***6000 Series Software Reference***.

CAUTION

Minimized Error Checking: 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 value is output.

Do not exceed update period: 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 controller, 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.

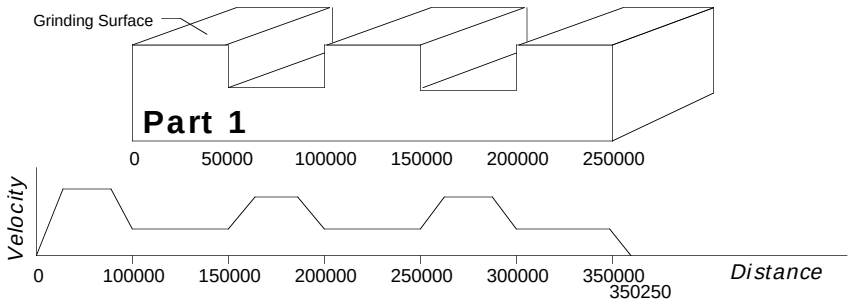
Executing a pause (P command or pause input): The occurrence of a pause command or a pause input will stop motion and disable the streaming mode.

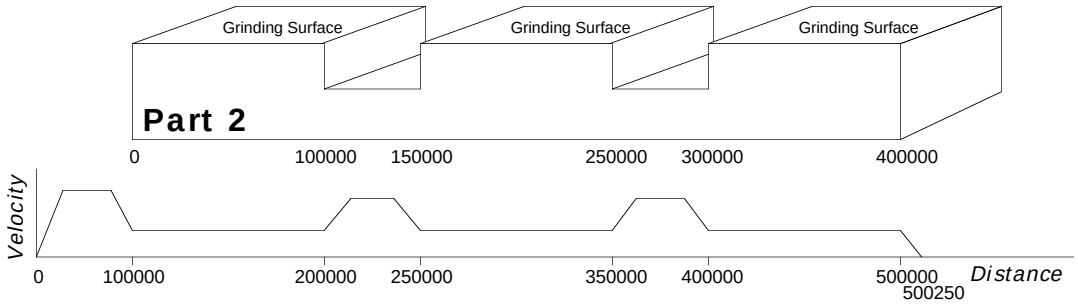
Keep the pulse width settings equal: All axes involved in the streaming mode (STREAM) must have the same PULSE setting (1 µs minimum).

Time-Distance Streaming Example

The example below illustrates a grinding application. The application involves two axes in an X-Y orientation. One axis controls the position of the grinding wheel relative to the part being ground (Y-axis). The other axis moves the grinding wheel parallel to the part being ground (X-axis). The motion on the Y-axis is accomplished with the standard motion commands (A, V, D, and GO). However, the motion on the X-axis is more complex, requiring time-distance streaming.

There are two parts that can be ground (see illustration below). The three faces labeled A, B, and C are to be ground at a speed of 2 rps, while traversing between each grind at 10 rps. The motion profiles for the X-axis of each part are shown.





Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The grinding wheel is rotated by a constant velocity motor that is either on or off. The motor is controlled by a single output. During the X-axis traverse, the constant velocity motor is turned on just prior to grinding, and turned off immediately after grinding. Points ON and OFF in the above illustration are where the motor is turned on and off.

The whole process will be initiated by an input, and will loop continuously until a second input is toggled.

GRINDING PROGRAM:

```
DEF MAIN           ; Begin definition of program main
PULSE1,1           ; Set pulse width to 1
DRES25000,25000    ; Drive resolution is 25000 steps/revolution
EOT32,32           ; Set end-of-transmission characters to spaces
LH0,0              ; Disable limits
MA11               ; Enable absolute mode on axes 1 & 2
                   ; (not applicable during streaming)
HOMLVL11           ; Set active high home level
HOM11              ; Go home on axes 1 and 2
A50,100            ; Set X and Y-axis acceleration
V10,10             ; Set X and Y-axis velocity
D0                 ; Set absolute distance for X-axis
VARS1="Part 1 or Part 2 ? "
REPEAT             ;
VAR1=READ1         ; Read a value into variable 1
WRITE"\13\10"      ;
UNTIL(VAR1=1 OR VAR1=2)  ; If variable 1 is not equal to 1 or 2,
                         ; repeat the above 3 commands
VAR1=VAR1*100+400000000  ; Variable 1 is used to determine grind length
REPEAT             ;
WRITE"Waiting for part to be loaded...\10\13"
WAIT(IN=b1)        ; Wait for input 1 to become active
WRITE"Part Loaded.\10\13\10\13"
D,175000           ; Y-axis distance
GO01               ; Move Y-axis into position to grind
GRIND              ; Start grinding on X-axis
D,0                ; Y-axis distance
GO01               ; Move Y-axis out of position
GO1                ; Move X-axis to home position
UNTIL(IN=bX1)      ; If input 2 is active, end repeat loop
WRITE"Finished Grinding for the Day\10\13"
END                ; End definition of program main

DEF GRIND          ; Begin definition of program grind
STD10              ; Set streaming update rate to 10 milliseconds
STREAM1            ; Enable time-distance streaming on axis 1
SD500              ; Begin traverse to grind point A
SD700
SD900
SD1100
SD1300
SD1500
SD1700
SD1900
SD2100
SD2300
SD400000030        ; Loop 30 times
SD2500             ; Move 2500 steps/update
SD500000000        ; End loop
SD210000000        ; Turn on output 1 to start the grinding wheel
SD2300             ; Decelerate to grind section A
SD2000
SD1800
SD1500
SD1200
SD900
SD700
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

GRINDING PROGRAM *(continued)*

```
SD600           ; Total distance moves after this SD point = 100000 steps
SD(VAR1)        ; Loop 100 times for Part 1, 200 times for Part 2
SD500           ; Move 500 steps/update (Grinding section A)
SD500000000     ; End loop
SD700           ; Begin traverse to grind point B
SD900
SD1100
SD1400
SD1700
SD2000
SD2200
SD400000012     ; Loop 12 times (distance = 30000 steps)
SD2500          ; Move 2500 steps/update
SD500000000     ; End loop
SD2200          ; Decelerate to grind section B
SD2000
SD1700
SD1400
SD1100
SD900
SD700
SD(VAR1)        ; Loop 100 times for Part 1, 200 times for Part 2
SD500           ; Move 500 steps/update (Grinding section B)
SD500000000     ; End loop
SD700           ; Begin traverse to grind point C
SD900
SD1100
SD1400
SD1700
SD2000
SD2200
SD400000012     ; Loop 12 times (distance = 30000 steps)
SD2500          ; Move 2500 steps/update
SD500000000     ; End loop
SD2200          ; Decelerate to grind section C
SD2000
SD1700
SD1400
SD1100
SD900
SD700
SD(VAR1)        ; Loop 100 times for Part 1, 200 times for Part 2
SD500           ; Move 500 steps/update (Grinding section C)
SD500000000     ; End loop
SD200000000     ; Turn off output 1 to stop the grinding wheel
SD250           ; Decelerate to a stop over two updates
SD0
SD700000000     ; Exit streaming mode
WAIT(MOV=b0)    ; Wait for motion to stop
END             ; End definition of program grind
;
; ******************************************************
; *  To initiate the program, execute the "main" command  *
; ******************************************************
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Linear Interpolation

NOTE

- *Linear Interpolation is not applicable to single-axis products.*
- *2-axis products can accommodate only 2-axis (X & Y) linear interpolation.*

The controller allows you to perform *linear interpolation*, the process of moving two or three orthogonal (right angle) linear axes to achieve linear (straight line) motion; a fourth axis may also participate in the move profile. The task is to derive appropriate move parameters to move from a current location to a new location, where each position is specified by a set of *Cartesian coordinates*. All axes must start, accelerate, decelerate, and stop in a synchronized manner.

The Initiate Linear Interpolated Motion (GOL) command initiates linear interpolation moves based on the parameters set with the D, PA, PAD, and PV commands. You simply enter the desired path acceleration (PA), the path deceleration (PAD), and the path velocity (PV) to arrive at the point in space (*end point*) specified with the distance (D) command; the controller internally calculates each axis' actual move profiles to achieve a straight-line path with these parameters.

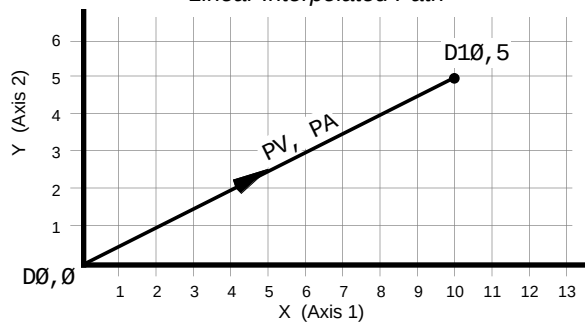
You can scale the acceleration, velocity, and distance with the PSCLA, PSCLV, and SCLD commands, respectively (see example below).

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

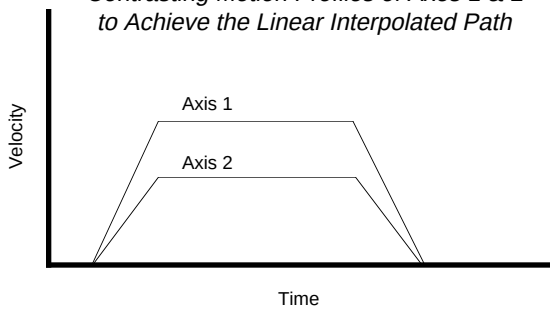
Code Sample

```
SCALE1          ; Enable scaling
@PSCLA4000      ; Set path acceleration scale factor to 4000 steps/unit
@PSCLV4000      ; Set path velocity scale factor to 4000 steps/unit
@SCLD4000       ; Set distance scale factor to 4000 step/unit
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           ; Set the distance to 10 & 5 units on axes 1 & 2, respectively
GOL11           ; Initiate linear interpolated motion on axes 1 & 2
                ; (see figure below)
```

Linear Interpolated Path



Contrasting Motion Profiles of Axes 1 & 2 to Achieve the Linear Interpolated Path



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Contouring (Circular Interpolation)

NOTE
- *Contouring is not applicable to single-axis products.*
- *2-axis products can accommodate only 2-axis (X & Y) contouring.*

6000 Series controllers allow you to define and execute two-dimensional motion paths. A *path* refers to the path traveled by the load in an X-Y plane, and must be defined before any motion takes place along that path. The X and Y axes can be specified as any of the controller's two axes.

Four-axis controllers: A third axis, labeled the C axis, may be included to keep an angular position which changes linearly with the path direction. The path direction is the vector addition of the travel of axes X and Y. A fourth axis labeled the P axis may be included to keep a position which is proportional to the distance traveled along the path described by X and Y. The X, Y, C and P axes can be specified as any of the controller's four axes.

A *path* consists of one or more line or arc segments whose end-points are specified in terms of X and Y positions. The end-point position specifications may be made using either absolute or incremental programming. The segments may be lines or arcs, both of which are described in greater detail in the following sections. Each path segment is determined by the end-point coordinates, and in the case of arcs, by the direction and radius or center. It is possible to accelerate, decelerate or travel at constant velocity (feedrate) during any type of segment, even between segments. For each segment, the user may also specify an output pattern which can be applied to the programmable outputs at the beginning of that segment.

All paths are continuous paths, (i.e., the motion will not stop between path segments, but must stop at the end of a path). It is not possible to define a path that stops motion within the path definition and then continues that path. To achieve this result, two individual paths must be defined and executed. A path may, however, be stopped and resumed by using a *Pause/Resume input* (see page 111) while the path is executing. In this case, motion will be decelerated and resumed along the path without loss of position. If one axis is stopped due to any other reason, the other axis will stop abruptly, and motion may not be resumed. Causes for motion being stopped may include encountering an end-of-travel limit, issuing a Kill (`!K`) command, detecting a stall (steppers), exceeding the max. allowable position error (servos), etc.

Path Definition

Contouring paths are defined like programs (using the `DEF` and `END` commands), but are compiled with the `PCOMP` command and executed with the `PRUN` command. Programs intended to be compiled as paths are stored in *Program* memory. After they are compiled with the `PCOMP` command, they remain in program memory and the *segments* (`PARCM`, `PARCOM`, `PARCOP`, `PARCP`, and `PLIN` statements) from the compiled profile are stored in *Compiled* memory. The `TDIR` command reports which programs are compiled as a compiled profile.

☞ *Compiled `GOBUF` segments are also stored in compiled memory (see page 13).*

The amount of RAM allocated for storing contouring path segments is determined by the `MEMORY` command setting. The table below identifies memory allocation defaults and limits for 6000 Series products. Further details on re-allocating memory are provided on page 12.

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 compiled profiles	1000,63000	1000,1499000	1000,39000	1000,149000	1000,149000
Max. # of compiled profiles	100	800	100	300	300
Max. # of compiled profile segments	875	20819	541	2069	2069

-M refers to the Expanded Memory Option

CAUTION

Issuing a memory allocation command (e.g., `MEMORY1000,39000`) will erase all existing programs and compiled contouring path segments. However, issuing the `MEMORY` command by itself (i.e., `MEMORY`—to request the status of how the memory is allocated) will not affect existing programs or path segments.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

You can store the maximum number of paths possible (see table above) as long as each path has at least one segment, and the sum of all the segments of all the paths does not exceed the controller's memory limitation for paths. All path definitions may be compiled and ready to execute at any time. Paths defined using 6000 commands are specified with a path name. Once a path definition is compiled, it may be executed repeatedly without being re-compiled.

Deleting (DEL) an existing path name will automatically delete the existing path compilation with that name. The PUCOMP command only deletes ("uncompiles") the path compilation, not the path program.

Example Code

In the example commands below, storage space is made available for the definition of path WID3 by first deleting the compiled version of paths WID1 and WID2. The DEF statement begins the definition of the path WID3.

```
PUCOMP WID1    ; Remove compilation of WID1
PUCOMP WID2    ; Remove compilation of WID2
DEF WID3       ; Begin definition of WID3
```

Participating Axes

NOTE

The mechanical resolution of all participating axes (identified with PAXES command) must be identical; scaling cannot compensate for mechanical variances. In addition, all participating axes must have the same PULSE settings and the same DRES settings (steppers only) or the same number of feedback device counts per unit of linear travel (servos only). If you change the PULSE setting, you will need to recompile (PCOMP) any previously compiled paths.

2-Axis Contouring

You can change the X-Y plane to an Y-X plane by using the PAXES command. A path definition default is PAXES1,2. For any path that uses axis 2 as the X axis and axis 1 as the Y axis, the path definition must start with PAXES2,1 (see example below)

Example Code

```
DEF DRAW1      ; Begin definition of DRAW1
PAXES2,1       ; Set contouring axes (axis #2 is X axis, axis #1 is Y axis)
```

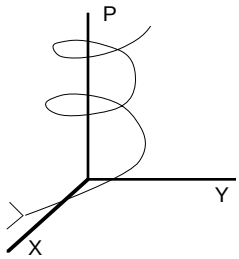
4-Axis Contouring

Some contouring applications may require the execution of more than one path to complete a part or finish an operation. The application may require that different paths take place in different planes of a three dimensional work area. In addition, some of the paths may require a third axis to move either tangent, or proportional to the path. For these reasons, a four-axis controller offers great flexibility in the specification of participating axes.

You may want to begin your path definition with the PAXES command; this will ensure that you have specified the appropriate axes to participate in the path. The X, Y, tangent (C) and proportional (P) axes can be specified as any of the four axes, and this specification must be made before any of the path travel specifications are made. The X and Y axes must be specified, the third (tangent) axis labeled C and the fourth (proportional) axis labeled P are optional.

The C axis will maintain an angular position which changes linearly with the direction of travel in the X-Y plane. This allows the C axis to control an object, which must stay tangent (or normal) to the direction of travel such as a cutting tool. The C axis must also be specified by its signed resolution. The magnitude of the resolution is the number of C axis motor steps in 360 degrees of an arc drawn by the X and Y axes. The sign of the resolution specifies the direction of rotation of the C axis. Refer to the PTAN command.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



The P 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 P axis to act as the Z axis in *helical interpolation* (see drawing at left), or to control the motion of any object which moves with distance and velocity proportional to the path. The P axis must also be specified by the signed ratio of P axis travel to path travel. The magnitude of this ratio may range from 0.001 to 1000. The sign of this ratio specifies the direction of rotation of the P axis. Refer to the PPRO command.

A sewing machine application may require all four axes (X,Y,C, and P). The X and Y axes would direct the sewing head along the required path. The C axis would keep the sewing head pointed into the direction of travel. The P axis would control the speed of the needle, so that an even stitch is made, regardless of path speed.

Example Code

The following example begins the definition of a path named DRAW1. The X and Y axes are specified to be axes 4 and 2. The path includes the C axis to be axis 1, with a resolution of 100,000 steps. It also includes the P axis to be axis 3, with a ratio of P axis travel to path travel specified as 2.5:1.

```
DEF DRAW1       ; Begin definition of DRAW1
PAXES4,2,1,3    ; Set contouring axes
PTAN100000      ; Define C axis resolution
PPRO2.5         ; Define P axis ratio
```

Path Acceleration, Deceleration, and Velocity

A path may be composed of many segments, each with their own motion parameters. The path velocity, acceleration, and deceleration specifications currently in effect at the time a segment is defined will apply to that segment. This allows construction of a path that moves at one velocity for a section of the path, then moves at a different velocity for another section.

In most cases, it will be desirable to maintain a constant velocity throughout the path, but it is easy to define a path in which each segment has its own velocity. For example, this may be useful when a tool needs to slow down to round a corner, or to allow the rate of glue application to be controlled by the path speed. Acceleration and deceleration may also be specified separately.

Example Code

The short code example below illustrates the specification of velocity, acceleration, and deceleration in that order.

```
SCALE1          ; Enable scaling
PSCLA25000      ; Scale path acceleration by 25,000
PSCLV20000      ; Scale path velocity by 20,000
PV0.5           ; Path velocity 10,000 counts/sec
PA16            ; Path acceleration 400,000 counts/sec/sec
PAD28           ; Path deceleration 700,000 counts/sec/sec
```

Segment End-point Coordinates

Steppers: All end-point position specifications are in units of motors steps, regardless of the current state of the ENC command.

The end-point position specifications of lines and arcs may be either absolute or incremental. The controller stores the end-point data for all of its compiled segments internally as incremental, relative to the start of the segment. But in order to ease the programming task, absolute coordinates and multiple coordinate systems may be used.

When incremental coordinates are used to specify an end-point, the X and Y end-point values represent the distances from the X and Y start point of the segment being specified. Center specifications of an arc are always incremental (i.e., relative to the start of that arc segment). When absolute coordinates are used to specify an end-point, the X and Y end-point values represent that segment's position in the specified coordinate system. Incremental and absolute programming are specified with the PAB command. Incremental programming is the default state at the beginning of a path definition.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Coordinate systems allow the assignment of an arbitrary X-Y position as a reference position for subsequent absolute end-point specifications. The controller allows the use of two coordinate systems for use with absolute coordinate programming. These are called the *Work* coordinate system and the *Local* coordinate system. These are specified with the PWC and PLC commands. Neither coordinate system needs to represent the actual absolute position of the axes when the path actually executes.

The Work and Local coordinate systems are provided to allow absolute end-point definition of a segment without needing to know the actual position of each axis when the segment is executed. If no PWC command precedes the first segment command when a path definition begins, the controller will place the start of the first segment at location (0,0) in the Work coordinate system. By using the PWC xpos,ypos command, the programmer defines subsequent absolute end-points to refer to the Work coordinate system, and also locates that coordinate system such that the starting position of the next segment is at (xpos, ypos) of the Work coordinate system.

The Local coordinate system is provided so that if a section of a path is to appear in multiple locations along the path, the segments that compose that section can be programmed in absolute coordinates. By using the PLC xpos,ypos command, the programmer defines subsequent absolute end-points to refer to the Local coordinate system, and also locates that coordinate system such that the starting position of the next segment is at (xpos, ypos) of the Local coordinate system.

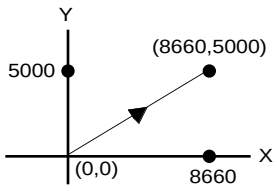
A single path definition may include both absolute and incremental programming, and be required to switch between Work and Local coordinates several times. At any point along a path definition, coordinates may be switched from absolute to incremental, or from incremental to absolute. When switching to absolute, all subsequent end-point specifications are assumed to be absolute with respect to the coordinate system in effect at that time. This remains true until the reference system is switched to incremental, or to a new absolute reference.

When switching from Work coordinates to Local coordinates, the Local X and Y start positions of the following segment must be specified with the PLC command. When starting a path definition with Work coordinates, or when switching to Work coordinates, the starting position of the next segment may either be specified or assumed. The controller toggles between the Work coordinate system and the Local coordinate system with the PL command.

Ease of programming results from the ability to switch between absolute and incremental, and to re-define the coordinate systems between sections of a path. This allows individual sections of path definition to have Local coordinate systems, yet still be integrated into the complete path.

Line Segments

Lines are the simpler of the two path segment types. The placement, length, and orientation of the line is completely specified by the end-point of the line segment and the end-point of the previous segment. As described above, end-points can be specified with absolute or incremental coordinates.

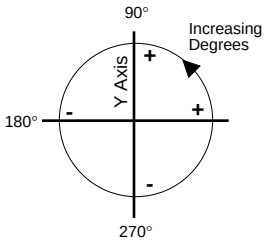


The example below is specified with incremental coordinates and results in a line segment 10,000 steps in length, at 30 degrees in the X-Y plane.

```
PLIN8660,5000    ; Line segment to (8660,5000) – see illustration at left
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Arc Segments



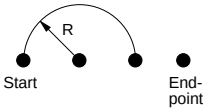
Arcs are more complex to specify than lines, because there are four possible ways to get from the start point to the end point. The radius of an arc may either be specified directly or implied by the center specification. In the controller, all path descriptions refer to the X-Y plane. The general convention describing the X-Y plane, as viewed from a drawing, is as follows. The X axis is shown as the left-right axis, with left being negative and right being positive. The Y axis is the up-down axis with down being negative and up being positive. Angles start at zero and increase in the CCW direction of rotation. A line segment, or the radius of an arc is at zero degrees if the incremental end-point has a positive X component and zero Y component. The angle is 90 degrees if the end-point has a positive Y component and zero X component.

Radius Tolerance Specifications

All arcs have an associated radius. In the controller, the radius may either be specified explicitly, or implied by a center specification. In both cases, it is possible that the radius may not be consistent with the specified end-point of the arc. This could be a result of improper specification, user calculation error, or of round-off error in the internal arithmetic of the controller. For this reason, the controller allows the specification of a radius tolerance (PRTOL). The radius tolerance is specified in the same units as the radius and X and Y data.

The radius tolerance has a factory default of ± one step, which is just enough to overcome round-off errors. The radius tolerance may be specified at any point along the path definition, and may be changed between one arc and the next. Each arc definition will be compared to the most recently specified radius tolerance. The radius tolerance should be about the same as the dimension tolerances of the finished product. The following paragraphs explain how the radius tolerance is used for the two types of arc specifications, and gives syntax examples for the radius tolerance specification.

Radius Specified Arcs

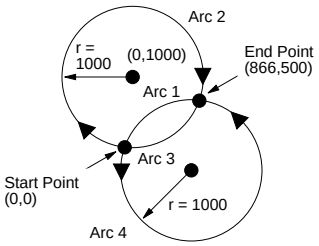


Specification of an arc using the radius method requires knowledge of the start point, the end point, and the sign and magnitude of the radius. The controller knows the start point to be either the start of the path, or the end of the previous segment. The end point and radius are provided by the user's program. It is possible to specify an impossible arc by specifying an end point that is more than twice the radius away from the start point (see drawing at left). In this case, the controller will automatically extend the radius to reach the end-point, provided that the automatic radius change does not exceed the user specified radius tolerance. If the required radius extension exceeds the radius tolerance, the controller will respond with an execution error, and no arc will be generated.

Example

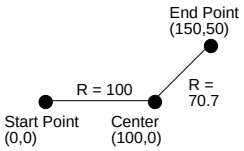
The following illustration shows the four possible ways to move from the start point to the end point using an arc of radius 1000. Arc 1 and 2 both travel in the CW direction, arc 3 and 4 both travel in the CCW direction. Arc 1 and 3 are both less than 180 degrees. An arc of 180 degrees or less is specified with a positive radius. Arc 2 and 4 are both greater than 180 degrees. An arc of more than 180 degrees is specified with a negative radius. The example code below shows the radius tolerance specification and the specifications of arcs 1, 2, 3, and 4 respectively.

```
SCALE1                ; Enable scaling
PSCLD100              ; Path position scaler
DEF arcs              ; Begin definition of path arcs
PRTOL5                ; 5 steps of radius tolerance
PARCP866,500,1000     ; Arc 1, CW < 180 degrees
PARCP866,500,-1000    ; Arc 2, CW > 180 degrees
PARCM866,500,1000     ; Arc 3, CCW < 180 degrees
PARCM866,500,-1000    ; Arc 4, CCW > 180 degrees
END                   ; End path definition
;*********************************************
;* To compile the arcs path, type "PCOMP arcs" *
;* To run the arcs path, type "PRUN arcs"      *
;*********************************************
```



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Center Specified Arcs



Specification of an arc using the center method requires knowledge of the start point, the end point, and the center point of the arc. The X coordinate of the center is referred to with the letter `I`, and the Y coordinate of the center is referred to with the letter `J`. When an arc is specified with the center, another potential problem arises.

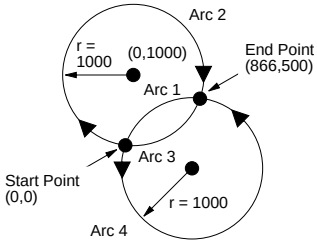
It is possible to specify the center of an arc such that the radius implied by the start point does not equal the radius implied by the end point (see illustration on left). In this case, the controller will re-locate the center so that the resulting arc has a uniform radius and the starting and ending angles come as close as possible to those implied by the user's center specification. This automatic center relocation will take place only if the start point and end point radius difference does not exceed the user specified radius tolerance. If the radius tolerance is exceeded, an execute error will result, and the arc will not be included in the path.

While automatic center relocation will ensure a continuous path, it may result in an abrupt change in path direction. This happens because a new location for the center results in a new tangent direction for an arc about that center.

Example

The example code below shows the specifications of arcs 1, 2, 3, and 4 for the drawing on the right. In the 6000 commands, the order of the data is X, Y, I, J from left to right.

```
SCALE1                  ; Enable scaling
PSCLD100                ; Path position scaler
DEF arcs2               ; Begin definition of path
PARCOP866,500,866,-500  ; Arc 1, CW < 180 degrees
PARCOP866,500,0,1000    ; Arc 2, CW > 180 degrees
PARCOM866,500,0,1000    ; Arc 3, CCW < 180 degrees
PARCOM866,500,866,-500  ; Arc 4, CCW > 180 degrees
END                     ; End path definition
;************************************************
;* To compile the arcs2 path, type "PCOMP arcs2" *
;* To run the arcs2 path, type "PRUN arcs2"      *
;************************************************
```

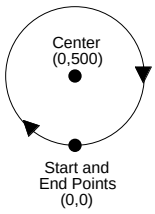


Circles

A circle is a special case of an arc whose end-point is the same as the starting point. Because these two points are the same, it is impossible to determine the location of the circle's center from a radius specification. For this reason, an arc that is a complete circle must be specified using the arc center specification method. An arc with identical starting and ending points specified with the radius method will be ignored. The circle shown below is specified with the example below.

Example

```
DEF circle          ; Begin definition of path circle
PARCOP0,0,0,500     ; Circle with center at (0,500)
END                 ; End path definition
;************************************************
;* To compile the circle path, type "PCOMP circle" *
;* To run the circle path, type "PRUN circle"      *
;************************************************
```



Segment Boundary

So far, all the examples given have shown isolated line or arc segments. Most paths will consist of many segments put together. The point at which the segments are connected is called a *segment boundary* in this text. The controller automatically ensures that the path is continuous, in that segments are placed end-to-end.

The path velocity may either be constant or change from segment to segment, according to user specification. Velocity changes use the specified acceleration and deceleration and may take place even across segment boundaries.

The programmer should ensure that direction of travel is also continuous across segment boundaries (see Figure A). If the direction change is abrupt (as shown in Figure B) the X and Y axes will suffer abrupt acceleration or deceleration. The controller ensures that there will be no abrupt direction change within a segment, but the programmer is responsible for ensuring that the direction is continuous across segment boundaries. At low speeds, some motor and

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

mechanical configurations will tolerate such abrupt changes, and the controller will accept such a program; however, it is generally good practice to design paths with smooth direction changes. This may be done by designing a path using arcs to round corners.

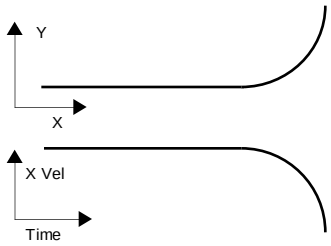


Figure A (Segment Example)

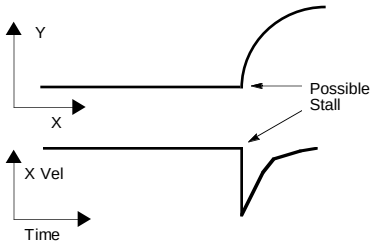


Figure B (Stall Example)

Using the C Axis *(4-axis products only)*

The *C axis* is an axis whose position changes in a manner linearly related to the direction of travel in X and Y (i.e., the path direction). The C axis would be used in applications that require a work piece or tool to remain tangent or perpendicular to the path direction. Examples would be: a knife always pointing into the cut, or a welding head staying normal to the weld.

The magnitude of the C axis resolution refers to the number of steps of C axis position change for 360 degrees of direction change in the X-Y plane. This number may be the same as, or different from the C axis motor resolution, allowing any gearing that is convenient for the mechanics. If the C axis load is to be driven directly, the C axis resolution should be the same as the C axis motor resolution. This will cause the C axis motor and load to rotate once when a circle is drawn by the X and Y axes. If the C axis load is to be geared (e.g., 5:1), the C axis resolution specifications should be five times the C axis motor resolution. This will cause the actual motor to rotate five times and the load to rotate once when a circle is drawn by the X and Y axes.

The number may be positive or negative, allowing greater flexibility in C axis motor mounting orientation. If the sign is positive, the C axis will rotate in the positive direction when CCW arcs are drawn. If the sign is negative, the C axis will rotate in the negative direction when counter-clockwise arcs are drawn.

The C axis is assumed to be in the proper position when path execution begins. It will change position only as the direction of travel changes. The program must position the C axis before the path is executed. This can be done with the `HOM` command or a `GO` to a position.

Because the C axis position changes linearly with the direction of X-Y travel, it is important to avoid path definitions which result in an abrupt direction change between segments. The segment boundary considerations for the C axis are similar to those for the X and Y axes, except that abrupt direction changes will result in abrupt C axis position changes. The X and Y axis would only suffer large accelerations, which may cause a stall in steppers or exceed the maximum position error (`SMPER` value) in servos. The C axis will suffer impossibly high velocity commands, causing stall and position loss in steppers or position error in servos.

Using the P Axis *(4-axis products only)*

The *P axis* is an axis whose position and velocity are proportional to the position and velocity traveled by the load along the path generated by X and Y. It can be used as the Z axis in helical interpolation, or to control other motion which must be proportional to the X-Y path motion. The proportionality of the P axis is specified as a ratio, with a range of ±0.001 to ±1000. The sign of the ratio determines which direction the motor will turn. The magnitude specifies the ratio of P axis travel to path travel, regardless of path direction or segment type. This ratio is essentially a position ratio, but because the ratio is maintained at every instant it also becomes a velocity ratio.

The P axis only responds to the distance traveled along the path, and is not affected by direction changes in the path. *The only caution that must be observed comes when a high ratio is specified. In this case, path velocity and acceleration are amplified, which may result in impossible velocities or stalls (steppers) or excessive position error (servos).*

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

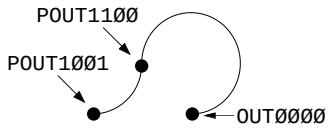
Outputs Along the Path

For each segment, you may also specify an output pattern (POUT), which is to be applied to the programmable outputs at the beginning of that segment and remains throughout that segment. These segment-defined output patterns are stored as part of the compiled path definition. These outputs will change state at some time between 1.5 ms before the beginning of the segment and 0.5 ms after the beginning of the segment. The programmable outputs may not be controlled more precisely than this, because the controller updates its path position every 2 ms for steppers and every system update for servos (see system update table in SSFR command description).

The path segment defined programmable outputs are provided so that plotting applications may raise and lower the pen, laser cutters may turn the laser on and off, glue applicators may be turned on and off, all at prescribed positions along the path. The output specification is stated before the segment definition, which holds that output state. In the example below, programmable outputs 2 and 4 are changed during the path segments.

Example Code

```
DEF prog1      ; Begin definition of path program prog1
POUT1001       ; Output pattern during first arc
PARCM5,5,5     ; Specify incremental X-Y endpoint position and
               ; radius arc <180° for 1/4 circle CCW arc
POUT1100       ; Output pattern during second arc
PARCM5,-5,-5   ; Specify incremental X-Y endpoint
               ; position and radius arc >180° for
               ; 3/4 circle CW arc
END            ; End definition of prog1
PCOMP prog1    ; Compile path program prog1
PRUN prog1     ; Execute path program prog1
OUT0000        ; Turn off programmable outputs 1-4
```



Paths Built Using 6000 Series Commands

When defining a given path, the commands that specify all of the path definitions must be contained in a named block defining that path. Each path definition block has a unique name that is used to distinguish one path from another. Because the path definition is stored as a program, many different paths may be stored, each defined with a unique name. A path definition block begins with a DEF command (containing its name) and ends with an END command.

The controller offers a command to compile (PCOMP) a named path definition block, and a separate command to execute (PRUN) a named path. Once a named path is compiled, it may be executed repeatedly without delay.

Compiling the Path

A PCOMP command will cause the controller to find the named path definition block and compile the path described by those commands, even if that pathname had been previously compiled. The use of variables (VAR) as parameters in path definition statements allow the same basic path to be re-defined with slightly different sizes and shapes. They may also be used to conditionally include or omit sections of the path.

Designed to allow compile-time determination of path parameters, there may be cases when the controller should prompt the operator or host computer for the value to be used for path velocity or segment end-points. Alternatively, these values may be read with the READ or DAT commands, allowing multiple calls of a single subroutine to define similar path sections with different data values. Commands that retrieve this data would be placed within the path definition, and would only prompt for the information when the path is compiled (e.g., PV(READ1)).

Example Code

```
VARS1="PATH VELOCITY ? "   ; Create message string
DEF path1                  ; Begin definition of path1
PAXES1,2                   ; Set path X & Y axes
PAB1                       ; Absolute path mode
PA100                      ; Path acceleration
PV(READ1)                  ; Path velocity, to be read in when compiling
PLIN25000,25000            ; Move in a line
PLIN(VAR2),(VAR3)          ; Move in a line, to be read in when compiling
END                        ; End definition of path1
PCOMP path1                ; Compile path1
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Executing the Path

A PRUN command will cause the controller to find the named path definition block and execute the path described by those commands, if that pathname has already been compiled (PCOMP).

The use of variables as parameters in the path definition statement is a method of allowing segment parameters to take new values each time the path is compiled. When the path is executing, the values of the variables do not affect the path parameters. If a change in a variable value is intended to affect the path parameters, that path must be re-compiled. The PRUN command performs the equivalent of a GOSUB to the named path definition block.

Possible Programming Errors

It is possible to create a situation in which the segment statements are interrupted. This could occur if an enabled ON condition becomes true. If an enabled ON condition (ONCOND) becomes true while running a compiled path, the branch to the ONP program will result. Motion from the path that was being executed will continue at the last segment velocity until it is stopped. Within the ONP program, a Stop command should be issued for all axes to stop the path from executing. For more information on program interrupts (ON conditions), see page 29.

Programming Examples

Figure A and Figure B show two simple paths that illustrate most of the controller segment types. For both figures, axis 1 is X and axis 2 is Y. The C and P axes are not included.

Figure A specifies the end-points with absolute coordinates. The default Work coordinate system with start point of (0,0) is used, so no PLØ statement is needed.

Figure B specifies the end-points with incremental coordinates. The state of the programmable outputs needs to be different for Handles than for Knobs. No other controller actions take place during these paths.

6000 Code

```
SCALE1              ; Enable scaling
PSCLA25000          ; Path acceleration scaler
PSCLV25000          ; Path velocity scaler
PSCLD25000          ; Path position scaler
DEF HANDLE          ; Begin HANDLE path definition
PAXES1,2            ; Set X axis as axis 1, and
                    ; set Y axis as axis 2
PAB1                ; Use absolute coordinates
POUT1100            ; Programmable pattern for
                    ; next segments
PARCOM10,10,0,10    ; CCW quarter circle
PLIN10,20           ; Vertical LINE segment
PARCP20,10,-10      ; CW 3/4 circle
PARCM20,0,5         ; CCW half circle
END                 ; End of HANDLE path definition
```

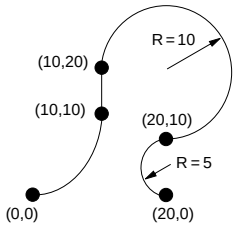


Figure A (HANDLE Example)

```
DEF KNOB            ; Begin KNOB path definition
PAXES1,2            ; Set X axis to be axis 1, and
                    ; set Y axis to be axis 2
PAB0                ; Use incremental coordinates
POUT0011            ; Programmable pattern for next
                    ; segments
PLIN30,0            ; Long LINE into circular knob
PARCOM0,0,0,10      ; CCW circle for the knob
PLIN10,0            ; Short LINE out of knob
END                 ; End of KNOB path definition
```

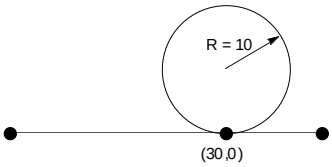


Figure B (KNOB Example)

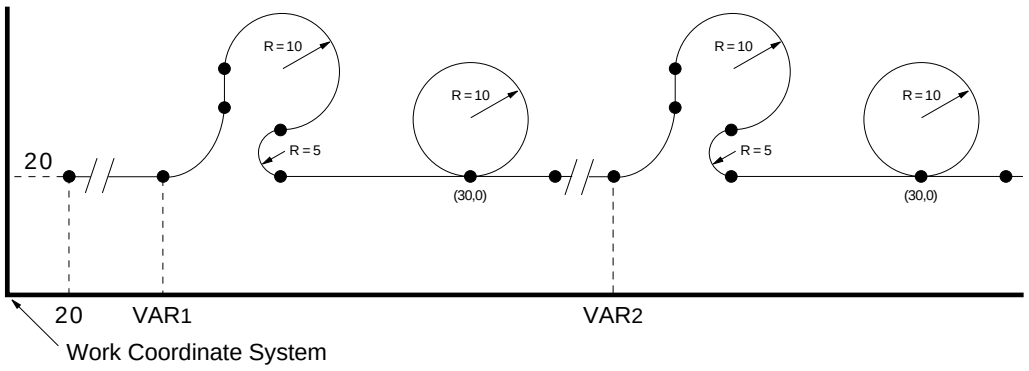
Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The third path consists of two pairs of the first two (see drawing below). Each pair is placed at variable locations within the Work coordinate system and the two pairs are connected with a Line segment. The Line leading into the first pair starts at (20,20) in the Work coordinate system. The first pair starts at (VAR1,20) and the second pair starts at (VAR2,20) in the Work coordinate system. HANDLE is defined using the Local coordinate system. Even though HANDLE is defined in absolute coordinates and appears in two different places along the path in PARTS, the statements describing it appear only once, in a path definition using local coordinates.

6000 Code

```
SCALE1              ; Enable scaling
PSCLA10000          ; Path acceleration scaler
PSCLV10000          ; Path velocity scaler
PSCLD10000          ; Path position scaler
DEF PARTS           ; Begin PARTS path definition
PAXES1,2            ; Set X axis to be axis 1, Y axis to be axis 2
PAB1                ; Use absolute coordinates
PWC20,20            ; Establish WORK coordinates
PL0                 ; Enable WORK coordinates
PLIN(VAR1),20       ; LINE to (VAR1,20)
PLC0,0              ; Specify LOCAL coordinate system
PL1                 ; Enable LOCAL coordinate system
PAB1                ; Use absolute coordinates
POUT1100            ; Programmable pattern for next segments
PARCOM10,10,0,10    ; CCW quarter circle
PLIN10,20           ; Vertical LINE segment
PARCP20,10,-10      ; CW 3/4 circle
PARCM20,0,5         ; CCW half circle
PAB0                ; Use incremental coordinates
POUT0011            ; Programmable pattern for next segments
PLIN30,0            ; Long LINE into circular knob
PARCOM0,0,0,10      ; CCW circle for the knob
PLIN10,0            ; Short LINE out of knob
PAB1                ; Use absolute coordinates
PL0                 ; Return to WORK coordinates
PLIN(VAR2),20       ; LINE to (VAR2,20)
PLC0,0              ; Specify LOCAL coordinate system
PL1                 ; Enable LOCAL coordinate system
PAB1                ; Use absolute coordinates
POUT1100            ; Programmable pattern for next segments
PARCOM10,10,0,10    ; CCW quarter circle
PLIN10,20           ; Vertical LINE segment
PARCP20,10,-10      ; CW 3/4 circle
PARCM20,0,5         ; CCW half circle
PAB0                ; Use incremental coordinates
POUT0011            ; Programmable pattern for next segments
PLIN30,0            ; Long LINE into circular knob
PARCOM0,0,0,10      ; CCW circle for the knob
PLIN10,0            ; Short LINE out of knob
END                 ; End of PARTS path definition

PCOMP PARTS         ; Compile PARTS path definition
```



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Compiled Motion Profiling

6000 Series products allow you to construct complex motion profiles for each individual axis. The profiles may contain:

- Sequences of motion
- Loops
- Programmable output changes
- Embedded dwells
- Direction changes
- Trigger functions

Related Commands:
Brief descriptions of related commands are found on page 169. For detailed descriptions, refer to the *6000 Series Software Reference*.

Compiled motion profiles are defined like programs (using the DEF and END commands); the commands used to construct the motion profile segments are stored in a program (stored in *Program* memory). This program is then compiled (using the PCOMP command) and the compiled profile *segments* (GOBUF, PLOOP, GOWHEN, TRGFN, POUTA, POUTB, POUTC, and POUTD statements) from the program are stored in *Compiled* memory. (**TIP**: The TDIR command reports which programs are compiled as a compiled profile.) You can then execute the compiled profile with the PRUN command.

☞ *Contouring path segments are also stored in compiled memory (see page 13).*

The amount of RAM allocated for storing compiled profile segments is determined by the MEMORY command setting. The table below identifies memory allocation defaults and limits for 6000 Series products. Further details on re-allocating memory are provided on page 12.

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 compiled profiles	1000,63000	1000,1499000	1000,39000	1000,149000	1000,149000
Max. # of compiled profiles	100	800	100	300	300
Max. # of compiled profile segments	875	20819	541	2069	2069

-M refers to the Expanded Memory Option

CAUTIONS

- Issuing a memory allocation command (e.g., MEMORY1ØØØ,39ØØØ) will erase all existing programs and compiled path segments. However, issuing the MEMORY command by itself (i.e., MEMORY—to request the status of how the memory is allocated) will not affect existing programs or segments.
- 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.

After compiling (PCOMP), you can execute the profiles with the PRUN command, and all of the motion and functions compiled into the profile are executed without any further commands during profile execution.

For multi-axis products, profiles on any combination of axes may be launched simultaneously with a single PRUN command. This provides a very powerful method of synchronizing the action of multiple axes with very simple programming. For example, in a four-axis product, one axis could be running a complex Following profile, while two other axes are contouring, and the fourth could be performing a multi-tiered velocity motion profile.

Because the motion and functions are *pre-compiled*, delays associated with command processing are eliminated during profile execution, allowing more rapid sequencing of actions than would be possible with programs which are not compiled. Command processing is then free to monitor other activities such as I/O and communications.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

NOTE

During compilation (PCOMP), most commands are executed the same as if no profile were being defined, even those which are not relevant to the construction of a profile. This is also true of non-compiled motion commands embedded in a compiled motion program during PCOMP. For this reason, it's good to limit commands between DEF and END to those which actually assist in the construction of the profile. Even for those that do actually assist in the construction of the profile, such as A, V, and D, it is important to remember that the command is executed and data actually changes, and it is not restored after compilation is completed.

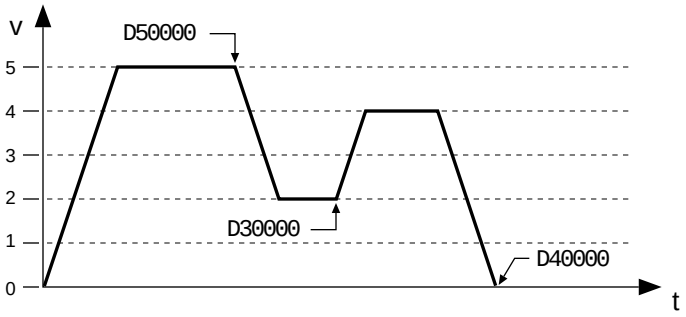
Each motion segment in a compiled motion profile may have its own distance, velocity, acceleration, and deceleration, as shown in the program example below:

Programming Example

```
DEF simple        ; Begin definition of program simple
MC0               ; Preset positioning mode (disable continuous mode)
MA0               ; Preset incremental positioning mode
D50000            ; Distance is 50000
A10               ; Acceleration is 10
AD10              ; Deceleration is 10
V5                ; Velocity is 5
GOBUF1            ; First motion segment for axis 1
D30000            ; Distance is 30000
V2                ; Velocity is 2
GOBUF1            ; Second motion segment for axis 1
D40000            ; Distance is 40000
V4                ; Velocity is 4
GOBUF1            ; Third motion segment for axis 1
END               ; End program definition

PCOMP simple      ; Compile simple
PRUN simple       ; Run simple
```

The resulting profile from the above program:



STATUS COMMANDS: Use these commands to check the status of compiled profiles.

System Status (TSSF, TSS, & SS commands):

- 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; this bit is 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: Reports the number of available segments in compiled memory.

TDIR: Identifies programs that are "compiled as a path" (compiled with the PCOMP command) and reports the percentage of remaining compiled memory.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Last Motion Segment Must End At Zero Velocity

When defining a profile, the last segment of motion in preset mode (MCØ) must end at zero velocity. When not using compiled loops (PLOOP), and when in preset mode (MCØ), the last GOBUF will automatically end at zero velocity. For example:

```
DEF P1    ; Begin definition of the P1 profile
MC0       ; Select preset positioning
MA0       ; Select incremental positioning
V1        ; Set velocity to 1 rps
          ; (assume no scaling)
D4000     ; Set distance to 4000
GOBUF1    ; First motion segment for axis 1
V2        ; Set velocity to 2 (second segment)
GOBUF1    ; Second motion segment
END       ; End definition of P1
PCOMP P1  ; Compile the P1 profile
```

The first motion segment of this profile consists of a 4000-step move at 1 rps. The second motion segment will ramp up to 2 rps, and finish a 4000-step move at zero velocity. The total distance moved with this profile is 8000 steps.

With compiled loops (PLOOP and PLN), the last segment within the loop must end at zero velocity or there must be a final segment placed outside the loop. Otherwise, an error ("ERROR: MOTION ENDS IN NON-ZERO VELOCITY–AXIS n") will be generated when you try to compile the program with the PCOMP command (see example below).

```
DEF P2    ; Begin definition of the P2 profile
MC0       ; Select preset positioning
MA0       ; Select incremental positioning
D4000     ; Set distance to 4000
PLOOP5    ; Loop (between PLOOP & PLN) 5 times
V1        ; Set velocity to 1 rps
GOBUF1    ; First motion segment for axis 1
PLN1      ; End loop
END       ; End definition of P2
PCOMP P2  ; Compile the P2 profile
```

This program will result in an error when it is compiled, because the last segment within the loop does not end in zero velocity.

To avoid the compile error, you could change the loop to 4 (PLOOP4) and include a final GOBUF1 command outside the loop (after the PLN1 command):

```
DEF P2    ; Begin definition of the P2 profile
MC0       ; Select preset positioning
MA0       ; Select incremental positioning
D4000     ; Set distance to 4000
PLOOP4    ; Loop (between PLOOP & PLN) 4 times
V1        ; Set velocity to 1 rps
GOBUF1    ; First motion segment for axis 1
PLN1      ; End loop
GOBUF1    ; Second motion segment for axis 1
END       ; End definition of P2
PCOMP P2  ; Compile the P2 profile
```

The 4000-step move will be repeated 4 times as part of the loop in the first motion segment. Because another GOBUF was added after the loop, the second motion segment (another 4000-step move) will end at zero velocity, as in the P1 profile above. The total distance moved with this profile is 20000 steps.

Another way to avoid the compilation error is to give the last segment in the loop a final velocity of zero (VFØ command). Note that in this programming example, each pass through the loop will end in a zero velocity.

```
DEF P3    ; Begin definition of the P3 profile
MC0       ; Select preset positioning
MA0       ; Select incremental positioning
D4000     ; Set distance to 4000
PLOOP5    ; Loop (between PLOOP & PLN) 5 times
V2        ; Set velocity to 2 rps
GOBUF1    ; First motion segment for axis 1
V1        ; Set velocity to 1 rps
VF0       ; Set final velocity to zero (0) rps
GOBUF1    ; Second motion segment for axis 1
PLN1      ; End loop
END       ; End definition of P3
PCOMP P3  ; Compile the P3 profile
```

The motion sequence (4000-step move at 2 rps ... 4000-step move at 1 rps ... stop) will be repeated 5 times. The total distance traveled will be 40000 steps.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Compiled Following Profiles

More details on Following are found in Chapter 6 (page 191).

The new FOLRNF command designates that the motor will move the load the distance designated in a preset GOBUF segment, completing the move at the specified final ratio. For the Revision 4.0 release of the 6000 series, the only allowable value for FOLRNF is zero (0). 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. FOLRNF is only useful with compiled preset Following moves because the starting and final ratios are already zero for motion initiated with GO.

Compiled motion profiles may be constructed with any combination of preset or continuous motion segments. A continuous (MC1) Following segment will start with the final ratio of the previous segment, and end with the ratio given by FOLRN and FOLRD. The motion segment will consist of one ramp from the starting ratio to the final ratio. Just as with continuous Following ramps outside of a compiled profile, the master travel over which the ramp takes place is specified with FOLMD. The slave travel over which the ramp takes place is simply the product of master travel and average ratio. Because the slave travel is not specified explicitly, it is possible for arithmetic round-off errors to cause actual slave travel during a ramp to differ from theoretical calculations. For applications in which slave distance is important, preset segments should be used.

A preset (MCØ) Following segment will also start with the final ratio of the previous segment, but may end in one of two ways. FOLRNF specifies the final ratio of a preset Following segment. As previously described, with this Revision 4.0 release, the only valid value for FOLRNF is zero (0). If FOLRNFØ is given before the GOBUF, the resulting motion segment will be constructed exactly as preset Following moves are outside of compiled profiles. In this case, the starting ratio must be zero, the final ratio will be zero, and the maximum intermediate ratio will be given by FOLRN and FOLRD. The relationships between ratio, master distance, and slave distance for this case are given on page 220 under the heading *Master and Slave Distance Calculations*. The FOLRNF command affects only the immediately subsequent preset Following segment, and must be given explicitly for each preset segment which is to end in zero ratio.

If FOLRNFØ is not given before the GOBUF, the segment will end with the ratio given by FOLRN and FOLRD, and need not start with zero ratio. This type of motion segment is constrained, however, to intermediate ratios which fall between the starting and final ratios.

Compiled profiles are built from motion segments created with the GOBUF command. For each individual axis, all motion segments in a compiled profile must use the same state of Following. That is, the motion segments may be all Following or all non-Following, but not a mixture of Following and non-Following (but at any point in time, separate axes may have different Following mode states).

The GOBUF command builds the appropriate type of motion segment based on the values of FOLMAS and FOLEN during compilation. These parameters may not be changed inside a compiled program after a GOBUF. The choice of zero or non-zero FOLMAS must be the same during PRUN as during PCOMP (if non-zero, the value can be changed, but still must be non-zero). If a non-zero FOLMAS is given, the value of FOLEN must be the same during PRUN as during PCOMP.

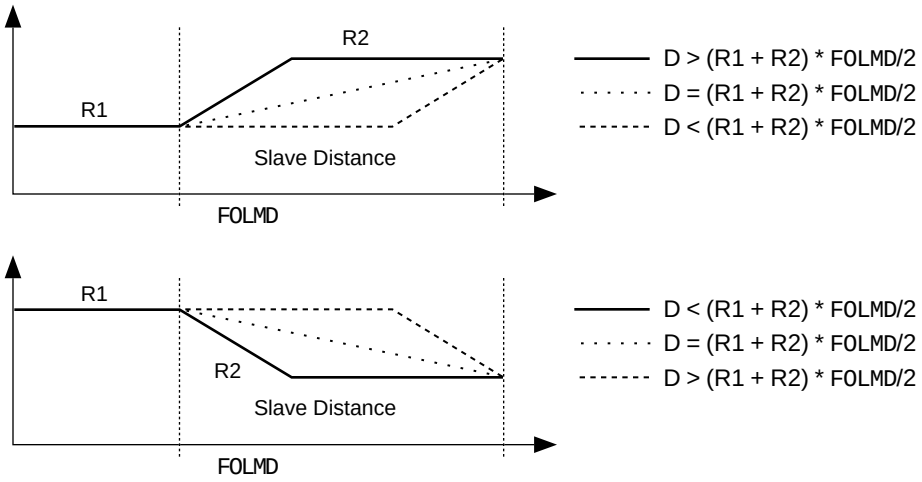
Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Distance Calculations For Compiled Following Moves

The graph below shows 6 possibilities of ratio change profiles for preset segments, with legal FOLMD and "D" values constrained by the requirement that the average ratio (given by "D"/FOLMD) is between R1 and R2. If the distance is outside these ranges, in the profile used to get from R1 to R2 over FOLMD (covering "D" slave distance), an error message will be generated during the PCOMP command. For the graphs shown, the constraints are expressed by:

(R1 * FOLMD) <= "D" <= (R2 * FOLMD) if R2 > R1

(R1 * FOLMD) >= "D" >= (R2 * FOLMD) if R2 < R1



The two graphs above show the cases of R1<R2 or R1>R2, but the distance calculations of the ramp and constant ratio portions are the same for the two cases. For each graph, the heavy lined profile (first case) of these mimics the shape of the corresponding preset velocity change (FOLENØ) segments in that the ramp takes place before the constant ratio portion. The second case occurs only if the distance specified exactly matches the start and end ratios and FOLMD1. In the third case, the ramp takes place after the constant ratio portion. In the first and third cases, only two segments are built, and the slave and master distances traveled in each segment are easily calculated with the simple formulas shown below. These formulas are based on positive ratios and master and slave distances. In the construction of Following profiles, ratios and master distances are always positive, with direction implied by the sign of the slave distance. For calculations with negative slave distances, simply use the magnitude of "D" in the formulas below, and invert the sign of the resulting slave distances.

Case 1 (Ramp first)	MD1 = [D-(R2*FOLMD)]/((R1-R2)/2)	where MD1 = master distance during ramp
	MD2 = FOLMD - MD1	where MD2 = master distance during flat
	D1 = .5*(R1+R2)*MD1	where D1 = slave distance during ramp
	D2 = D - D1	where D2 = slave distance during flat
Case 2 (Ramp only)	MD1 = FOLMD	where MD1 = master distance during ramp
	D1 = D	where D1 = slave distance during ramp
Case 3 (Ramp last)	MD1 = [D-(R1*FOLMD)]/((R2-R1)/2)	where MD1 = master distance during ramp
	MD2 = FOLMD - MD1	where MD2 = master distance during flat
	D1 = .5*(R1+R2)*MD1	where D1 = slave distance during ramp
	D2 = D - D1	where D2 = slave distance during flat

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Dwells and Direction Changes

Compiled profiles may incorporate changes in direction only if the preceding motion segment has come to rest. This may be achieved for non-Following segments either by creating a continuous segment with a goal velocity of zero, or by preceding a preset segment with VFØ. It may be achieved for Following segments either by creating a continuous or preset segment with a goal ratio of zero, or by preceding a preset segment with FOLRNFØ. In all cases, motion within the profile comes to rest, although the profile is not yet complete. Even though the motor is not moving, the axis status bit 1 (AS.1) will remain set, indicating a profile is still underway. Only then can you change direction (using the D+ or D- command, D~ is not allowed) within a profile. An attempt to incorporate changes in direction if the preceding motion segment has not come to rest will result in a compilation error.

In many applications, it may be useful to create a time delay between moves. For example, a machine cycle may require a move out, dwell for 2 seconds, and move back. To create this dwell, a compiled GOWHEN may be used between the two moves. The code within a compiled program may look like:

```
MC0             ; Preset positioning used
MA0             ; Incremental positioning used
D(VAR1)         ; Target position is in VAR1
VF0             ; Motion comes to rest at end of move
GOBUF1          ; Create move out segment
GOWHEN(T=2000)  ; Profile delays for 2 seconds
D-              ; Return position is home (direction reversed)
VF0             ; Motion comes to rest at end of move
GOBUF1          ; Create move back home segment
```

In Following applications, it may be more useful to create a master travel delay between moves. For example, a machine cycle replacing a cam may require a move out, dwell for 2000 master counts, and move back. To create this dwell, a compiled GOBUF of zero slave distance may be used between the two moves. The code within a compiled program may look like:

```
MC0             ; Preset positioning used
MA0             ; Incremental positioning used
D(VAR1)         ; Target position is in VAR1
FOLMD4000       ; Move takes place over 4000 master counts
FOLRNF0         ; Motion comes to rest at end of move
GOBUF1          ; Create move out segment
D0              ; No change in target position
FOLMD2000       ; Dwell takes place over 2000 master counts
FOLRNF0         ; Motion comes to rest at end of  “move” (dwell)
GOBUF1          ; Create dwell segment
D(VAR1)         ; Return position is home (direction change implied)
D-              ; Return position is home (direction reversed)
FOLMD4000       ; Move takes place over 4000 master counts
FOLRNF0         ; Motion comes to rest at end of move
GOBUF1          ; Create move back home segment
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Compiled Motion Versus On-The-Fly Motion

The two basic ways of creating a complex profile are with compiled motion or with on-the-fly pre-emptive GO commands. With compiled motion, portions of a profile are built piece by piece, and stored for later execution. Compiled motion is appropriate for profiles with motion segments of pre-determined velocity, acceleration and distance. Compiled motion profiles allow for shorter motion segments, which results in faster cycle times because there is no command processing and execution delay. The axes may perform their own motion control and coordination, freeing program flow for other tasks, such as I/O, machine control, and host requests. The disadvantages to pre-defined compiled motion profiles are the amount of memory use and limited run-time decision making and I/O processing.

With pre-emptive GO moves, the motion profile underway is pre-empted with a new profile when a new GO command is issued. The new GO command 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.

The table below summarizes the differences between the use of compiled motion and on-the-fly motion.

Command/Issue	Compiled Motion	On-The-Fly Motion
GOBUF	Constructs motion segment and appends to previously constructed segment	N/A
PRUN	Used to launch previously compiled motion	N/A
GO	GO causes move during PCOMP	GO Constructs & launches profile, even if moving
Direction changes	Only if previous motion segment comes to rest (MCØ & VFØ or MC1 & VØ), else compile error	Not allowed during motion, else AS.3Ø, ER.1Ø
Insufficient room for AD (decel) value	Same as on-the-fly	Decel is modified (steppers); Motion is killed, AS.3Ø (servos)

Related Commands

GOBUF — *Store a Motion Segment in Compiled Memory*:

The 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. An individual axis profile is constructed by sequentially appending motion segments using GOBUF commands. Each motion segment may have its own distance to travel, velocity, acceleration, and deceleration.

The end of a GOBUF motion segment in preset mode is determined by the distance or position specified. The end of a GOBUF motion segment in continuous mode is determined by the goal velocity specified. In both cases, the final velocity and position achieved by a segment will be the starting velocity and position for the next segment. If either type of segment is followed by a GOWHEN command, the segment's final velocity will be maintained until the GOWHEN condition becomes true.

PLOOP & PLN — *Loop Start & Loop End (Compiled Motion only)*:

The PLOOP and PLN commands specify the beginning and end of an axis-specific profile loop, respectively. All segments defined between the PLOOP and PLN commands are included within that loop.

VF & FOLRNF — *Final Velocity & Numerator of Slave-to-Master Final Ratio*:

The VF and FOLRNF commands are used to designate that the motor will move the load the distance designated in a preset GOBUF motion segment, completing the move at a final speed of zero. The VF command is used when the Following mode is disabled (FOLENØ). The FOLRNF command is used when the Following mode is enabled (FOLEN1).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

GOWHEN — *Conditional GO*:

The GOWHEN command has been modified to allow use in compiled motion profiles. Now, when GOWHEN is compiled in a profile, the GOWHEN condition is stored as part of that profile instead of being executed immediately. When progress through the profile reaches the compiled GOWHEN, AS.26 is set, and the next segment's execution will be suspended until the GOWHEN condition becomes true. This allows subsequent GOWHEN and GOBUF combinations to be issued and stored, instead of overriding each other.

TRGFN — *Trigger Functions*:

The TRGFN command has been modified to allow use in compiled motion profiles. Now, when TRGFN is compiled in a profile, the TRGFN condition is stored as part of that profile instead of being executed immediately. When progress through the profile reaches the compiled TRGFN, the embedded trigger functions are assigned to that trigger. AS.26 is set if the GOWHEN function has been assigned to the trigger, and the next segment's execution will be suspended until the specified trigger input goes active. This allows subsequent TRGFN, GOWHEN, and GOBUF combinations to be issued and stored, instead of overriding each other.

PCOMP, PRUN & PUCOMP — *Compile a Program, Run a Compiled Program, & Un-Compile a Compiled Program*:

The PCOMP, PRUN, and PUCOMP commands have been modified to incorporate individual axis profiles within compiled motion profiles. Compiled motion for the 6000 series now allows the user to construct complex motion programs using an individual contour (a series of arcs and lines), individual axis profiles (a series of GOBUF commands), or a path (combination of contours and individual axis profiles).

POUTA, POUTB, POUTC, POUTD — *Output During Compiled Motion Profile — Axes 1, 2, 3 & 4:*

The POUTA, POUTB, POUTC, and POUTD commands turn the programmable output bits on and off for axes 1, 2, 3 and 4, respectively.

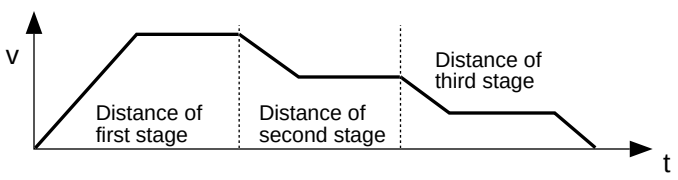
TSEG & SEG — Transfer/Display (TSEG) or Assign (SEG) the Number of Free Segment Buffers:

The TSEG command returns the number of free segment buffers in compiled memory. The SEG command is used to assign the number of free segment buffers in compiled memory to a variable or to make a comparison against another value.

Compiled Motion — Sample Application 1

A manufacturer has an application where wire is being wrapped onto a spindle. There is a motor controlling the rotational speed of the spindle. Every application of the spindle requires that the motor runs at a fast speed with a slow acceleration for the first few revolutions, a medium speed for the next couple of revolutions, and a slower speed as the spindle gets fuller to maintain somewhat of a constant velocity off the feed wire. The technician would like to use an RP240 to enter the velocity and number of revolutions for each stage of winding. Programmable outputs 1, 2 and 3 are wired to status LEDs, and should go on for the respective stages of winding (output 1 for stage 1, etc.).

Profile



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Program

```
DEF PROFIL              ; Define motion profile program
VAR10 = 4000 * VAR4     ; Get distance of first stage
                        ; (assuming 4000 steps/revolution)
D(VAR10)                ; Set distance
V(VAR1)                 ; Set velocity of first stage
POUTA.1-1               ; Turn output 1 on
GOBUF1                  ; Build motion
VAR10 = 4000 * VAR5     ; Get distance of second stage
D(VAR10)                ; Set distance
V(VAR2)                 ; Set velocity of second stage
POUTA01                 ; Turn output 1 off and output 2 on
GOBUF1                  ; Build motion
VAR10 = 4000 * VAR6     ; Get distance of third stage
D(VAR10)                ; Set distance
V(VAR3)                 ; Set velocity of third stage
POUTAx01                ; Turn output 2 off and output 3 on
GOBUF1                  ; Build motion
POUTA.3-0               ; Turn off output 3
END                     ; End motion profile program

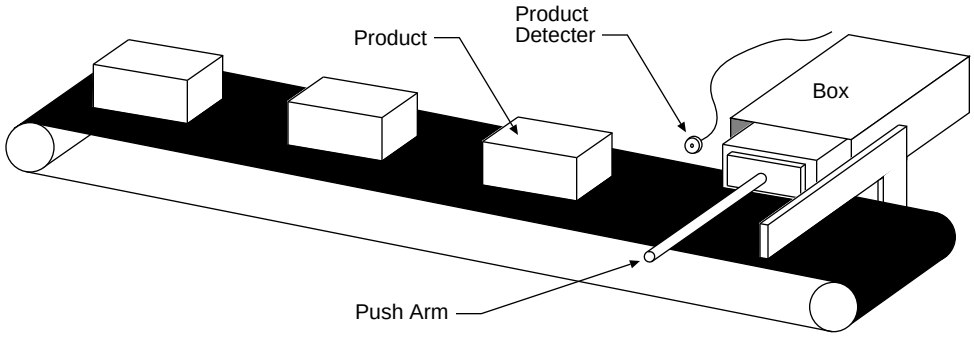

DEF EXMPL1              ; Define program example 1
L                       ; Continual loop of program execution
DCLEAR0                 ; Clear all lines on RP240 display
DPCUR1,1                ; Position cursor at line 1, column 1
DWRITE"ENTER VELOCITY STAGE 1"    ; Prompt user
VAR1 = DREAD            ; Get 1st velocity from RP240 entry
DCLEAR1                 ; Clear line 1 on RP240 display
DPCUR1,1                ; Position cursor at line 1, column 1
DWRITE"ENTER VELOCITY STAGE 2"    ; Prompt user
VAR2 = DREAD            ; Get 2nd velocity from RP240 entry
DCLEAR1                 ; Clear line 1 on RP240 display
DPCUR1,1                ; Position cursor at line 1, column 1
DWRITE"ENTER VELOCITY STAGE 3"    ; Prompt user
VAR3 = DREAD            ; Get 3rd velocity from RP240 entry
DCLEAR1                 ; Clear line 1 on RP240 display
DPCUR1,1                ; Position cursor at line 1, column 1
DWRITE"ENTER REVOLUTIONS STAGE 1" ; Prompt user
VAR4 = DREAD            ; Get # of windings 1st stage from RP240 entry
DCLEAR1                 ; Clear line 1 on RP240 display
DPCUR1,1                ; Position cursor at line 1, column 1
DWRITE"ENTER REVOLUTIONS STAGE 2" ; Prompt user
VAR5 = DREAD            ; Get # of windings 2nd stage from RP240 entry
DCLEAR1                 ; Clear line 1 on RP240 display
DPCUR1,1                ; Position cursor at line 1, column 1
DWRITE”ENTER REVOLUTIONS STAGE 3" ; Prompt user
VAR6 = DREAD            ; Get # of windings 3rd stage from RP240 entry
PCOMP PROFIL            ; Re-compile profile with new vel/dist info
$AGAIN                  ; Label for repeating same profile
PRUN PROFIL             ; Execute profile
DCLEAR1                 ; Clear line 1 on RP240 display
DPCUR1,1                ; Position cursor at line 1, column 1
DWRITE"SAME DATA (1=YES,2=NO)"  ; Prompt user if perform again with old data
VAR7 = DREAD            ; Get response
IF(VAR7=1)              ; If user wants to perform same profile
  GOTO AGAIN            ; perform again
NIF                     ; End conditional
LN                      ; End command execution loop
END                     ; End definition program example 1

; ****************************************
; * To begin, execute the EXMPL1 program   *
; ****************************************
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

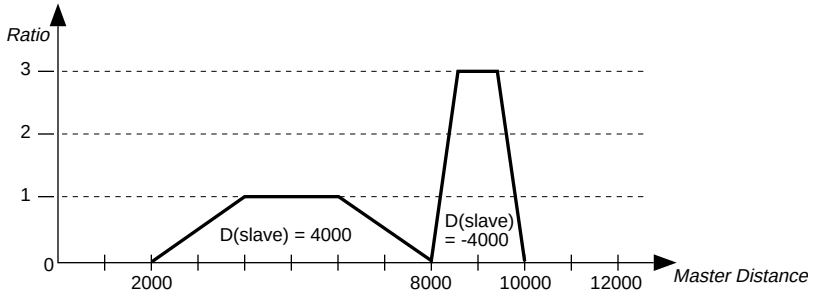
Compiled Motion — Sample Application 2

Here's an example of replacing a mechanical cam using a compiled Following profile. There is evenly spaced product coming in on a feeder belt. The feeder belt may vary in speed. The cam that you are replacing controls a push arm that will push the product into a box for shipping. You would also like the arm to retract at a faster rate than it extends. In other words, you would like to have a smooth push to load and a fast retract to set up for the next product. Since this is a cam, this profile must repeat continuously for each product or master cycle but won't start until the first product is detected.



The feeder belt is the master and the master cycle length (space from the front of one product to the front of the next) is 12000 master (encoder) counts on the feeder belt. The push of the product will start 2000 counts into the master cycle. The push will take place over 6000 master counts, and the retract over 2000 master counts. The distance the push arm (slave) must travel is 4000 counts. Assume the detector is wired to trigger A. Below is a graph of this Following profile.

Profile



Program

```
; Setup code
FOLMAS21          ; Master is coming in on encoder 2
FOLEN1            ; Enable Following
INFNC17-H         ; Assuming input 17 is trigger A (1 axis products)
INFEN1            ; Enable inputs

; Motion program
DEL EXPL2         ; Delete program (in case it already exists in memory)
DEF EXPL2         ; Begin definition of program example 2
TRGFNA1           ; Launch profile upon receiving trig A
                  ; (1st product detected)
PLOOP0            ; Loop continuously to mimic a mechanical cam

; Program first move - dwell
FOLRN1            ; Set up ratios - numerator
FOLRD1            ; and denominator
FOLMD2000         ; Over a distance of 2000 master steps
D0                ; slave will not move
FOLRNF0           ; and end at zero ratio
GOBUF1            ; Build motion
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
; Program second move - positive slave move
FOLMD6000          ; Over a distance of 6000 master steps
D4000              ; slave will move 4000 steps
FOLRNF0            ; and end at zero ratio
GOBUF1             ; Build motion

; Program third move - negative slave move
FOLRN3             ; New ratio to accommodate larger distance of slave travel
FOLMD2000          ; Over a distance of 2000 master steps
D-4000             ; slave will move -4000 steps
FOLRNF0            ; and end at zero ratio
GOBUF1             ; Build motion

; Program last move - dwell
FOLMD2000          ; Over a distance of 2000 master steps
D0                 ; slave will not move
FOLRNF0            ; and end at zero ratio
GOBUF1             ; Build motion
PLN1               ; Close cam loop
END                ; End program example 2

PCOMP EXPL2        ; Compile program EXPL2

; **  To execute the program, enter the PRUN EXPL2 command  **
```

Program Modification

Let's now modify the constraints of the system. Let's say that the product will be spaced roughly 12000 master counts apart. It may or may not be exactly 12000, but it will never be less than 10000 (just to make sure the retraction finishes before the next product is detected). We can then modify the program to wait for the product to be detected each cycle. We can also take the extra "dwell" or zero distance move out of the end of the profile. See program below:

```
; Setup code
FOLMAS21           ; Master is coming in on encoder 2
FOLEN1             ; Enable Following mode
INFNC17-H          ; Assuming input 17 is trigger A (1 axis products)
INFEN1             ; Enable inputs

; Motion program
DEL EXPL2B         ; Delete program (in case it already exists in memory)
DEF EXPL2B         ; Begin definition of program example 2b
PLOOP0             ; Loop continuously to mimic a mechanical cam
TRGFNA1            ; Pause profile until trig A received (detect next product)

; Program first move - dwell
FOLRN1             ; Set up ratios - numerator
FOLRD1             ; and denominator
FOLMD2000          ; Over a distance of 2000 master steps
D0                 ; slave will not move
FOLRNF0            ; and end at zero ratio
GOBUF1             ; Build motion

; Program second move - positive slave move
FOLMD6000          ; Over a distance of 6000 master steps
D4000              ; slave will move 4000 steps
FOLRNF0            ; and end at zero ratio
GOBUF1             ; Build motion

; Program third move - negative slave move
FOLRN3             ; New ratio to accommodate larger distance of slave travel
FOLMD2000          ; Over a distance of 2000 master steps
D-4000             ; slave will move -4000 steps
FOLRNF0            ; and end at zero ratio
GOBUF1             ; Build motion
PLN1               ; Close cam loop
END                ; End program example 2b

PCOMP EXPL2B       ; Compile program EXPL2B

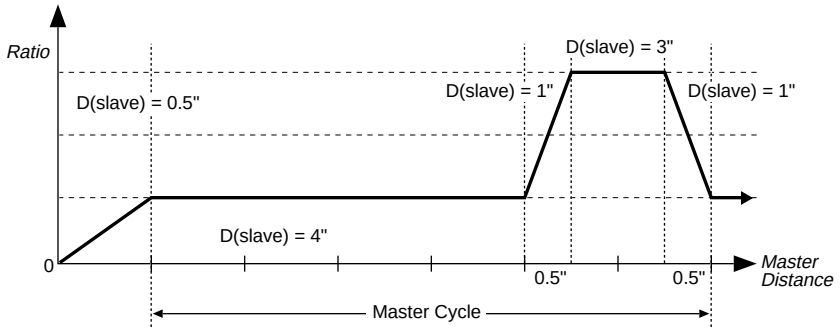
; ********************************************************
; * To execute the program, enter the PRUN EXPL2B command  *
; ********************************************************
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Compiled Motion — Sample Application 3

In this application, there is a wheel that stamps a logo onto the product. The product is assumed to be entering at a constant and fixed spacing, each product is 4 inches in length with 2 inches separating each unit. The stamp wheel has a circumference of 9 inches, and must be traveling at a 1 to 1 ratio with the product at the time of stamping. The stamp wheel must then travel five inches in just 2 inches of master travel. There is a sensor wired to trigger A of the 6000 controller to detect the first product and start the cycling. At the time of the trigger the product is 1 inch away from contact with the stamp wheel. Assume that the home position of the slave is 0.5" away from a stamp. The mechanics of the system give 3000 steps of master travel per inch and 1500 steps of slave travel per inch.

Profile



As you can see above, we have a multi-tiered Following profile. By multi-tiered we mean that ratio is changing from a non-zero value to another non-zero value. To program this profile effectively, we will break the profile into pieces as shown with the dotted lines in the above illustration:

Program

```
FOLMAS21            ; Define the master as encoder on axis 2
FOLEN1              ; Enable Following
INFEN1              ; Enable input functions
INFNC25-H           ; Trigger interrupt on trigger A
SCLMAS3000          ; Set scaling of master steps per inch
SCLD1500            ; Set scaling of slave steps per inch
SCALE1              ; Enable scaling

DEF EXMPL3          ; Start definition of example program 3
TRGFNA1             ; Launch profile when trigger A occurs

; Program first ramp from ratio 0 to ratio 1
FOLRD1              ; Set Following ratio - denominator
FOLRN1              ; Set the Following ratio at 1 to 1
FOLMD1              ; Over a master distance of 1"
D0.5                ; Slave will travel 0.5"
GOBUF1              ; Build motion

PLOOP0              ; Start the continuous loop

; Program constant ratio
FOLRN1              ; At a 1 to 1 ratio
FOLMD4              ; Over a master distance of 4"
D4                  ; Slave will travel 4"
GOBUF1              ; Build motion

; Program ramp to new ratio
FOLRN3              ; Go to a 3 to 1 ratio
FOLMD0.5            ; Over a master distance of 0.5"
D1                  ; Slave will travel 1"
GOBUF1              ; Build motion
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
; Program second constant ratio
FOLRN3              ; At a 3 to 1 ratio
FOLMD1              ; Over a master distance of 1″
D3                  ; Slave will travel 3″
GOBUF1              ; Build motion

; Program ramp to lower ratio
FOLRN1              ; Go to a 1 to 1 ratio
FOLMD0.5            ; Over a master distance of 0.5″
D1                  ; Slave will travel 1″
GOBUF1              ; Build motion
PLN1                ; Close motion loop

; Define the exit motion
FOLRN0              ; Stop slave at zero ratio (and zero velocity)
FOLMD1              ; Over a master distance of 1″
D0.5                ; And a slave distance of 0.5″
GOBUF1              ; Build motion
END                 ; End definition of example program 3

PCOMP EXMPL3        ; Compile example program 3

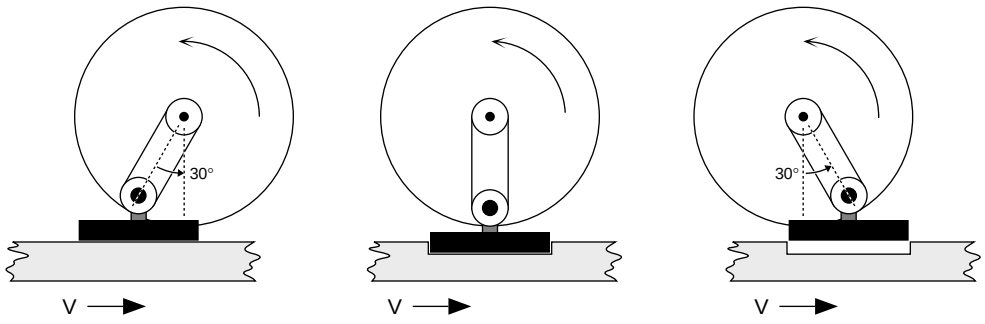
; ********************************************************
; * To execute the program, enter the PRUN EXMPL3 command   *
; ********************************************************
```

NOTE: The `GOBUF` command has been added to the "`Define the exit motion`" portion of the program despite the fact that an infinite loop has been programmed earlier in the program. This is to avoid an error message when the program is compiled.

Compiled Motion — Sample Application 4

A manufacturer of stamped molds needs to make a machine which will stamp molds into a continuous flow of extruded plastic material. The stamp must be lowered 0.5 inches into the plastic to leave the correct impression. Because the flow is continuous, the stamp must also move in synchronization with plastic in the direction of flow as it is lowered and raised. The initial design approach to the machine required two axes of motion. One was needed to lower and raise the stamp, the other to allow the stamp to follow the plastic. With the availability of complex Following cam profiles the job can done with a single axis.

In the drawing below, the stamp is attached to a rotating arm in such a way that the stamp remains level as the arm rotates. The length of the arm at the stamp fixture, or radius of rotation, is exactly one inch. The arm is mounted above the plastic so that at the bottom of its rotation (270 degrees), the stamp will be 0.5 inches into the plastic. Using trigonometry, the horizontal and vertical positions and speeds may be calculated at other arm angles. Because the stamp must follow the flow of the plastic, we must adjust the ratio of rotational speed to plastic speed so that the horizontal velocity component of the arm stays at 1:1 with the plastic while the stamp is in the plastic.



drawings are not to scale

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The table below shows these relationships. The arm is directly driven with a servo motor having 4096 steps per revolution. The table shows increments of 30 degrees, which is about 341 servo motor steps, or about .524 slave inches measured around the circumference described by rotation of the arm. The plastic flow is measured with an encoder giving 1000 steps per inch of flow. To maintain ratios in terms of inches, FOLRD will always be 1000. The required FOLRN value is simply the inverse of the arm's horizontal velocity component multiplied by the number of slave steps per inch. The corresponding ratio in terms of surface speeds is given in parentheses. The required FOLMD is the number of master steps corresponding to the horizontal component of slave rotation.

Arm angle, degrees	Horizontal component (in.) = cos(deg)	FOLMD = 1000 * delta cos(deg)	Horizontal vel component = -sin(deg)	Required FOLRN = -651.9/sin(deg)
210	-0.866	n/a	0.500	1304 (2:1)
240	-0.500	366	0.866	753 (1.155:1)
270	0.000	500	1.000	652 (1:1)
300	0.500	500	0.866	753 (1.155:1)
330	0.866	366	0.500	1304 (2:1)

The profile that we construct from these number is meant to approximate the inverse sine function in the last column, but of course, will actually be a series of ramps and constant ratio segments. Let's review the Compiled Following Move Distance Calculations to determines the exact shape and error in the first motion segment(from 210 to 240 degrees). First, we need to determine if the ramp or constant ratio is first for that segment. Using ratios and distances in inches, we have:

```
R1 = 2                  ;starting ratio
R2 = 1.155              ;final ratio
D = (2*pi)/12 = .524    ;distance at stamp hinge
FOLMD=.366              ;travel along plastic
```

We find (R1+R2) * FOLMD/2 = .577, which is greater than D, so the "Ramp First" equations apply to this segment. Let's examine the error at the junction between the ramp and constant ratio portion of this segment.

```
MD1 = [D - (R2 * FOLMD)] / ( (R1 - R2) / 2) = 0.239 master inches
D1 = 0.5 * (R1 + R2) * MD1 = 0.377 slave inches at circumference = 21.6 degrees
cos(210+21.6) - cos(210) = -0.621 - (-0.866) = 0.245 inches slave horizontal travel
error = horizontal slave travel - master travel = 0.245 -  0.239 = 0.006 inches
```

A similar calculation may be done for the "elbow" of the next of the next segment, and symmetry indicates these errors will be the same between 270 and 330 degrees. The error along intermediate points may be found with linear interpolation of ratio and master distance. In this case, the errors fall within manufacturing tolerance. If the errors were too large, the travel could be broken into more segments, each with exactly correct positions and ratios at their boundaries.

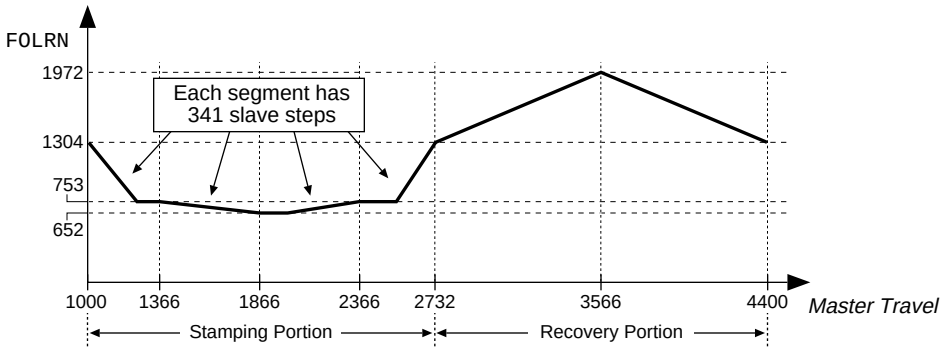
So far, we have only discussed the portion of the profile which lowers and raises the stamp. During the remainder of the profile, the arm must continue its rotation to bring the stamp to its starting position in time for the next mold. The mold is 3 inches long, and .4 inches are needed between molds for strength at the edges. This makes the total master cycle 3.4 inches long. The total slave cycle must be 4096 steps, so the segments required to bring the arm around must complete the portions of master and slave cycles not already accounted for. We will create two segments, which divide the remaining master and slave travels in two, and are mirror images of each other. The average ratio of these two segments must simply be slave travel divided by master travel, i.e., (D / FOLMD). As previously determined, the FOLRN value for the boundaries of the stamping portion of the profile is 638. From this value and the average ratio, we can calculate the peak FOLRN value.

```
D = 0.5 * remaining slave = 0.5 * (4096 - 4 * 341) = 1366
FOLMD = 0.5 * remaining master = 0.5 * [1000 * (3.4 - 2 * 0.866)] = 834
peak ratio = FOLRN/1000
0.5 * (FOLRN/1000 + 1304/1000) = average ratio = D / FOLMD = 1366 / 834 = 1.638
FOLRN = 1972   (solved from above)
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Finally, we need to design a segment used to create a smooth entry into the repetitive portion of the profile. We'll assume that the home position of the arm is at 180 degrees, so it needs to achieve the FOLRN ratio of 1304 in 30 degrees (341 slave steps). Using the same averaging arithmetic as above, the required master distance for the entry segment is 523 steps. A sensor is positioned with this entry segment in mind, and wired to **TRG-A**. A function to start motion when the sensor is triggered will be imbedded inside the profile. The motion segments for the stamping portion and recovery portions of the profile must be enclosed in a loop, and may be programmed by picking the numbers from the table and equations above. Because the ratio denominators are the same for all segments, and the slave distances are the same for the entry and each of the stamping segments, these are commanded only when the values change.

Repetitive Portion of Profile



Program

```
FOLMAS21          ; Follow extra encoder
FOLEN1            ; Enable Following mode
INFEN1            ; Enable input functions
INFNC17-H         ; Enable trigger A for interrupt
SCALE0            ; Parameters are in steps

DEF STAMP         ; Start program definition
TRGFNA1           ; Profile starts upon trigger A
FOLRD1000         ; Ratio denominator, 1000 steps per inch

;define the entry segment
D341              ; Distance of 341 steps is about 30 degrees
FOLRN1304         ; Goal ratio for start segment
FOLMD523          ; Master distance during ramp
GOBUF1            ; Build start segment

PLOOP0            ; Start the continuous loop

;this profile section starts 2-to-1 ratio, or a starting FOLRN1304
FOLRN753          ; Goal ratio for segment
FOLMD366          ; Master travel in steps for segment
GOBUF1            ; Build motion segment

;the 2nd section of profile starts with the final ratio of the 1st section
FOLRN652          ; Goal ratio for segment
FOLMD500          ; Master travel in steps for segment
GOBUF1            ; Build motion segment

;the next two sections are mirror images of the first two
FOLRN753          ; Goal ratio for third segment
FOLMD500          ; Master travel in steps for 3rd segment
GOBUF1            ; Build motion segment
FOLRN1304         ; Goal ratio for 4th segment
FOLMD366          ; Master travel in steps for 4th segment
GOBUF1            ; Build motion segment

;the next two sections complete the loop and are mirror images of each other
D1366             ; Slave travel in recovery segments
FOLMD834          ; Master travel in steps for recovery segments
FOLRN1972         ; Goal ratio for ramp up segment
GOBUF1            ; Build ramp up motion segment
FOLRN523          ; Goal ratio for ramp down segment
GOBUF1            ; Build ramp down motion segment
PLN1              ; End of loop cycle

;finally, a segment to end motion
D341              ; Distance of 341 steps is about 30 degrees
FOLRN0            ; Goal ratio for end segment
FOLMD1000         ; Master distance during ramp
GOBUF1            ; Build end segment
END               ; End of STAMP program definition

PCOMP STAMP       ; Compile the program

; *********************************************************
; * To execute the program, enter the PRUN STAMP command   *
; *********************************************************
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

On-the-Fly Motion (pre-emptive GOs)

While motion is in progress, 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) followed by the GO command. If the continuous command execution mode is enabled (COMEXC1), you can execute buffered commands; otherwise (COMEXCØ), you must prefix each command with an immediate command identifier (e.g., !A, !V, !D, !MC, 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). On-the-fly motion changes are applicable only for motion started with the GO command, and not for motion started with other commands such as HOM, JOG, JOY, PRUN or GOL.

On-the-fly motion changes are most likely to be used to change the velocity and/or goal position of a preset move already underway. In the event that the goal position is completely unknown before motion starts, a move may be started in continuous mode (MC1), with a switch to preset mode (MCØ), a distance command (D), and a GO given later. In absolute positioning mode (MA1) the new goal position given with a pre-emptive GO is explicit in the D command. In incremental positioning (MAØ) the distance given with a new pre-emptive GO is always measured from the at-rest position before the original GO. If a move is stopped (with the S command), and then resumed (with the C command), this resumed motion is considered to be part of the original GO. A subsequent distance given with a new pre-emptive GO is measured from the at rest position before the original GO, not the intermediate stopped position.

Programming Example: *This program creates a 2-tiered profile (single-axis) that changes velocity and deceleration at specific motor positions.*

```
SCALE0           ; Disable scaling
DEL OTF          ; Delete program (in case program is already in memory)
DEF OTF          ; Begin definition of program
PSET0            ; Set position to zero
SCALE0           ; disable scaling
COMEXC1          ; Enable continuous command processing mode
MC0              ; Select preset positioning
MA0              ; Select incremental positioning
A20              ; Set accel to 20 revs/sec/sec
AD20             ; Set decel to 20 revs/sec/sec
V9               ; Set velocity to 9 revs/sec
D500000          ; Set distance to 20 revs
GO1              ; Initiate motion
WAIT(PM>100000)  ; Wait until the motor position is > 100000 steps (4 revs)
V4               ; Slow down for machine operation
GO1              ; Initiate new profile with new velocity
WAIT(PM>450000)  ; Wait until the motor position is > 450000 steps (18 revs)
AD5              ; Set decel for gentle stop
V1               ; Slow down for gentle stop
GO1              ; Initiate new profile with new velocity
END              ; End program definition
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The table below summarizes the restrictions on pre-emptive GOs.

Condition	Possible?
Execute GO during MC1 & FOLENØ	Yes
Execute GO during MC1 & FOLEN1	Yes
Execute GO during MCØ & FOLENØ	Yes
Execute GO during MCØ & FOLEN1	No
Change MC setting during motion	Yes (but cannot change MC1 to MCØ during FOLEN1)
Change ENC setting during motion	No
Change FOLENØ to FOLEN1 during motion	No
Change FOLEN1 to FOLENØ during motion	Only while MC1, constant ratio, and not shifting

OTF Error Conditions

Further instructions about handling error conditions are provided on page 30.

The ability to change the goal position on the fly raises the possibility that the new position goal of an on-the-fly GO cannot be reached with the current direction, velocity, and deceleration. If this happens, an error condition is flagged in axis status bit #30 (AS.3Ø) for that axis and in error status bit #10 (ER.1Ø).

If the direction of the new goal position is opposite that of current travel, the 6000 controller will kill motion (stop motion abruptly) and set AS.3Ø and ER.1Ø.

If there has not yet been an overshoot, but it is not possible to decelerate to the new distance from the current velocity using the specified AD value, the action taken by stepper controllers is different from that taken by servo controllers:

- With steppers, the deceleration will be modified to avoid overshoot (as shown in *Scenario #2* below) and the AS.3Ø and ER.1Ø status bits will not be set.
- With servos, this case is considered an overshoot, the controller with kill the move, and the AS.3Ø and ER.1Ø status bits will be set.

RELATED STATUS COMMANDS

Axis Status — Bit #30: (this status bit is cleared with the next GO command)

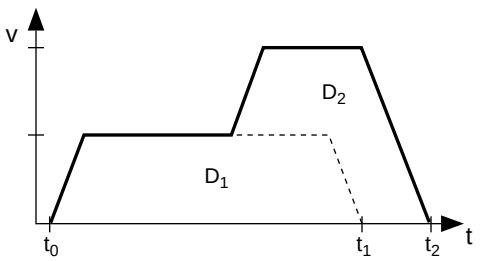
AS.3Ø..Assignment & comparison operator — use in a conditional expression (see pg. 25).
TASF....Full text description of each status bit. (see "Preset Move Overshot" line item)
TAS ..... Binary report of each status bit (bits 1-32 from left to right). See bit #30.

Error Status — Bit #10: The error status is monitored and reported only if you enable error-checking bit #10 with the ERROR command (e.g., ERROR.1Ø-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.1Ø..Assignment & comparison operator — use in a conditional expression (see pg. 25).
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.

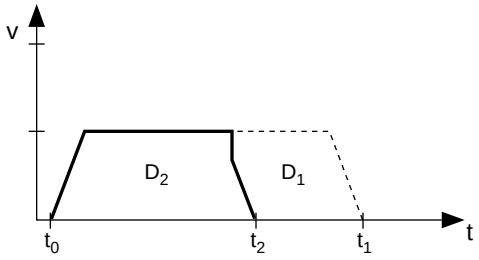
Error Condition Scenarios

Scenario #1: OTF change of velocity and distance, where new commanded distance (D_2) is greater than the original distance (D_1) that was pre-empted [$D_2 > D_1$]. The distances are the areas under the profiles, starting at t_0 for both. If the original move had continued, D_1 would have been reached at time t_1. D_2 is reached at time t_2.

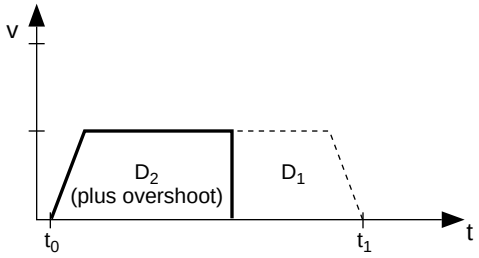


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

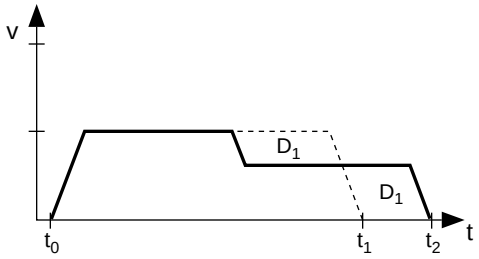
Scenario #2: OTF change of distance, where new commanded distance (D_2) is less than the original distance (D_1) that was pre-empted [$D_2 < D_1$]. In this example, D_2 is beyond the position where the OTF change was entered, however D_2 can not be reached with the commanded deceleration. In steppers, an instantaneous velocity change is made such that D_2 can be reached with the commanded deceleration (in servos, motion is killed if the new destination cannot be reached with the commanded deceleration).



Scenario #3: OTF change of distance, where new commanded distance (D_2) is less than the original distance (D_1) that was pre-empted [$D_2 < D_1$]. In this example, the position where the OTF change was entered is already beyond D_2. In the event of an overshoot (steppers only), motion is killed and `AS.3Ø` and `ER.1Ø` bits are set.



Scenario #4: OTF change of velocity. Note that motion must continue for a longer time at the reduced velocity to reach the original commanded distance than if it had continued at the original velocity ($t_2 > t_1$).

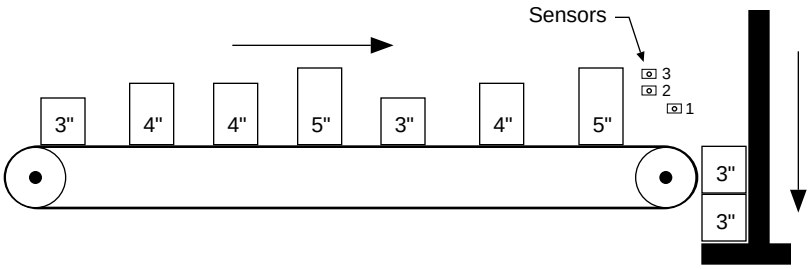


On-The-Fly Motion — Sample Application

A manufacturer of three products wishes to produce a “sampler-pak” package which will contain a few of each of his products. The products all have the same width and length, but are 3, 4, and 5 inches high respectively. The 3 products are fed from individual lines into a common conveyor, and arrive at a stacking and wrapping station. At this station, a tray accepts a product and must have moved down by that product’s height by the time the next product arrives. This means that each time a new product arrives, the velocity of the tray must be changed to match the height of that product. Although product spacing will be regular, the ordering of product type on the common conveyor will be random, due to variations in the input lines. Also, a finished sampler-pak should contain 5 products or be at least 18 inches high, whichever occurs first. This means that the total move distance of the tray will be unknown until the last product arrives. When the last product is stacked, an output is asserted which will pause the conveyor and start the wrapping process. When wrapping is complete, the sampler-pak is removed from the tray, and the tray returns to the starting position.

The basic problems in this application are that the move distance is not known until near the end, and the velocity must change on the fly. As the products approach the tray, they are detected with a near vertical arrangement of three sensors. Products of heights 3, 4, and 5 inches are detected by 1, 2, or all 3 sensors respectively. Input 1 always detects a product, and switches last, so that the others will be stable. When each product is identified, the motion profile is modified accordingly.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



Program (portion only)

```
VAR1=0                          ; Initialize product count
VAR2=0                          ; Initialize move distance variable
VAR3=0                          ; Initialize velocity
A10                             ; Moderate acceleration
MC0                             ; Start with preset move
WHILE(VAR1<5 AND VAR2<18)       ; Loop until cycle complete
WAIT(IN.1=B1)                   ; Wait for start of next product
VAR1=VAR1+1                     ; Update product count
IF(IN.2=B1)                     ; If not a 3" product
  IF(IN.3=B1)                   ; If it is a 5" product
    VAR3=5                      ; Set velocity
    VAR2=VAR2+5                 ; Update distance
  ELSE                          ; If not 5", must be 4"
    VAR3=4                      ; Set velocity
    VAR2=VAR2+4                 ; Update distance
  NIF                           ; End of 5" case check
ELSE                            ; 3" inch case
  VAR3=3                        ; Set velocity
  VAR2=VAR2+3                   ; Update distance
NIF                             ; End of 3" case check
V(VAR3)                         ; New velocity
D(VAR2)                         ; New distance
WAIT(IN.1=B0)                   ; Wait for end of this product
GO1                             ; Implement new distance and velocity
NWHILE                          ; Sampler-pak completed product detection
WAIT(AS.1=B0)                   ; Wait for move to complete
OUT1                            ; Output to indicate stacking complete
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Registration

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.

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

There are several methods of preventing unwanted registration moves:

- 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 page 109 or 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 *Sample Application 2* below.

- 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 *Sample Application 3* below.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

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 (see pg. 25).
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.3Ø ..... Assignment & comparison operator — use in a conditional expression (see pg. 25).
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.

Further instructions about handling error conditions are provided on page 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.1Ø-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.1Ø ..... Assignment & comparison operator — use in a conditional expression (see pg. 25).
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 (see page 112), the respective system status bit is cleared (set to Ø).

SS ..... Assignment & comparison operator — use in a conditional expression (see pg. 25). 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.

How to Set up a Registration Move

Before you can initiate a registration move, you must program these elements (refer also to the programming examples below):

- 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 112). NOTE: When configured as Trigger Interrupts, the triggers cannot be affected by the input enable (INEN) command.
- 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.

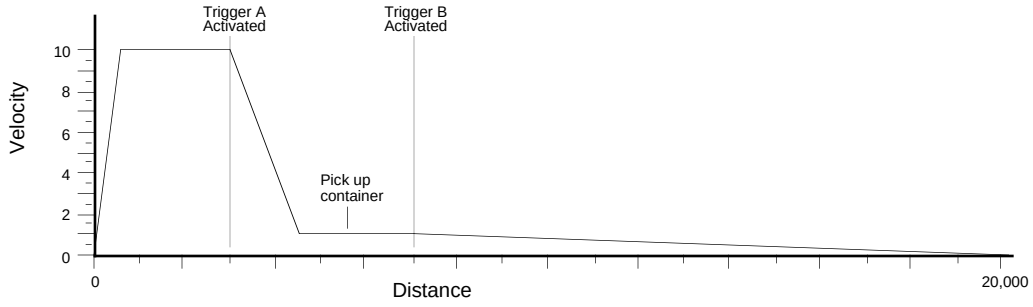
  NOTE: The registration move is executed using the A, AA, AD, ADA, and V values that were in effect when the REG command was entered. Stepper products: The position captured (motor or encoder) and the positioning mode (motor steps or encoder steps) used for registration are determined upon the ENC command setting in effect when the registration move was first defined with the REG command.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Registration – Sample Application 1

In this example, two-tiered registration is achieved (see illustration below). While axes 1 is executing it's 50,000-unit move, 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 conveyor belts. 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 slow the load to gentle stop.

```
DEL REGI1     ; Delete program (in case program already resides in memory)
DEF REGI1     ; Begin program definition
INFNC25-H     ; Define input #25 (trigger A) as a trigger interrupt input
INFNC26-H     ; Define input #26 (trigger B) as a trigger interrupt input
INFEN1        ; Enable programmable input functions defined with INFNC command
ENC0x         ; Set positioning mode to motor step mode (FOR STEPPERS ONLY)
A20           ; Set acceleration on axis 1 to 20 units/sec2
AD40          ; Set deceleration on axis 1 to 40 units/sec2
V1            ; Set velocity on axis 1 to 1 unit/sec
REGA4000      ; Set trigger A's registration distance on axis 1 to 4000 units
              ; (registration A move will use the ENC, A, AD, & V values above)
ENC0x         ; Set positioning mode to motor step mode (FOR STEPPERS ONLY)
A5            ; Set acceleration on axis 1 to 5 units/sec/sec
AD2           ; Set deceleration on axis 1 to 2 units/sec/sec
V.5           ; Set velocity on axis 1 to 0.5 units/sec
REGB13000     ; Set trigger B's registration distance on axis 1 to 13,000 units
              ; (registration B move will use the ENC, A, AD, & V values above)
RE10          ; Enable registration on axis 1 only
A50           ; Set acceleration to 50 units/sec/sec on axis 1
AD50          ; Set deceleration to 50 units/sec/sec on axis 1
V10           ; Set velocity to 10 unit/sec on axis 1
D50000        ; Set distance to 50000 units on axis 1
GO10          ; Initiate motion on axis 1
END           ; End program definition
```

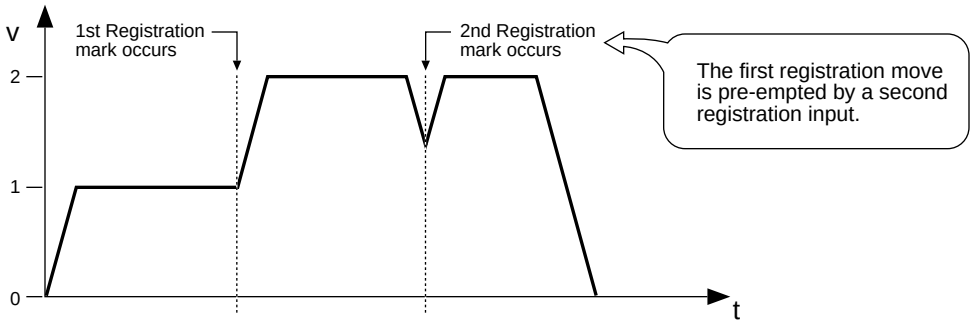


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Registration – Sample Application 2

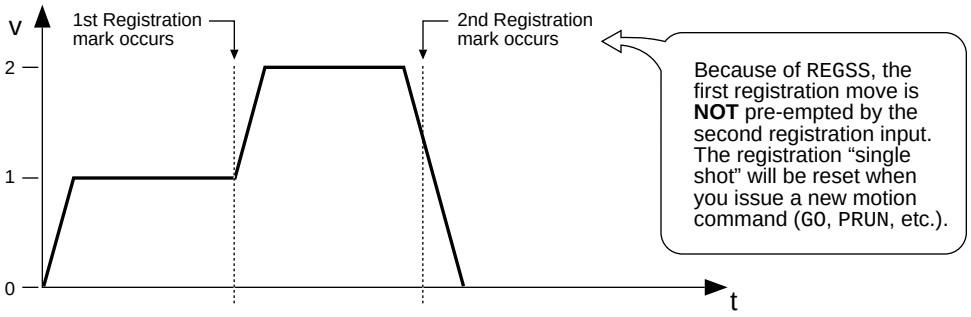
A user has a line of material with randomly spaced registration marks. It is known that the first mark must initiate a registration move, and that each registration move cannot be interrupted or the end product will be destroyed. Since the distance between marks is random, it is impossible to predict if a second registration mark will occur before the first registration move has finished.

```
DEL REGI2     ; Delete program (in case program already resides in memory)
DEF REGI2     ; Begin program definition
INFEN1        ; Enable input functions
INFNC25-H     ; Trigger capture mode for trigger A (input #25)
RE1           ; Enable registration
V2            ; Set registration move to a velocity of 2 rps
AD.5          ; a deceleration of 0.5 rev/sec/sec
REGA20000     ; and a distance of 20000 steps
MC1           ; Start a mode continuous
V1            ; move at a velocity of 1 rps
GO1           ; Initiate motion
END           ; End program definition
```



In order to stop the second registration from occurring, REGSS can be used:

```
DEL REGI2b    ; Delete program (in case program already resides in memory)
DEF REGI2b    ; Begin program definition
INFEN1        ; Enable input functions
INFNC25-H     ; Trigger capture mode for trigger A (input #25)
RE1           ; Enable registration
V2            ; Set registration move to a velocity of 2 rps
REGA20000     ; and a distance of 20000 steps
REGSS1        ; Enable registration single shot mode
MC1           ; Start a mode continuous
V1            ; move at a velocity of 1 rps
GO1           ; Initiate motion
END           ; End program definition
```

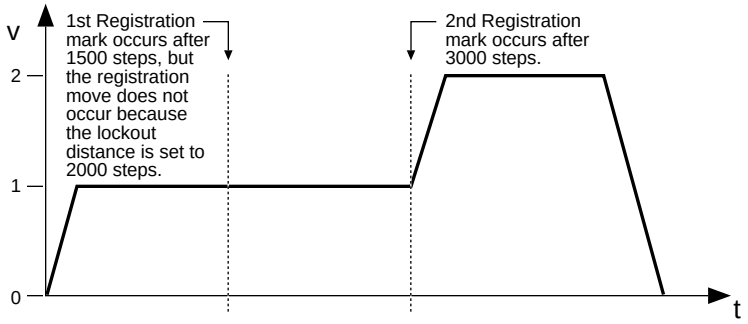


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Registration – Sample Application 3

A print wheel uses registration to initiate each print cycle. From the beginning of motion, the controller should ignore all registration marks before traveling 2000 steps. This is to ensure that the unit is up to speed and that the registration mark is a valid one.

```
DEL REGI3     ; Delete program (in case program already resides in memory)
DEF REGI3     ; Begin program definition
INFEN1        ; Enable input functions
INFNC25-H     ; Trigger capture mode for trigger A (input #25)
RE1           ; Enable registration
V2            ; Set registration move to a velocity of 2 rps
REGA2500      ; and a distance of 2500 steps
REGLOD2000    ; Set registration lockout distance to 2000 steps
MC1           ; Start a mode continuous
V1            ; move at a velocity of 1 rps
GO1           ; Initiate motion
END           ; End program definition
```



Synchronizing Motion (GOWHEN and TRGFN operations)

GOWHEN and TRGFN allow you to synchronize the execution of motion on one or more axes:

- GOWHEN — synchronize execution of the subsequent start-motion command (GO, GOL, FSHFC, or FSHFD) to:
  - Position (commanded, feedback device, motor, master, slave, Following shift)
  - Master cycle number
  - Input status
  - Time delay (dwell)
- TRGFN:
  - Suspend execution of the next start-motion command (GO, GOL, FSHFC, or FSHFD) until the specified trigger input goes active.
  - Suspend beginning a new Following master cycle until the specified trigger input goes active.

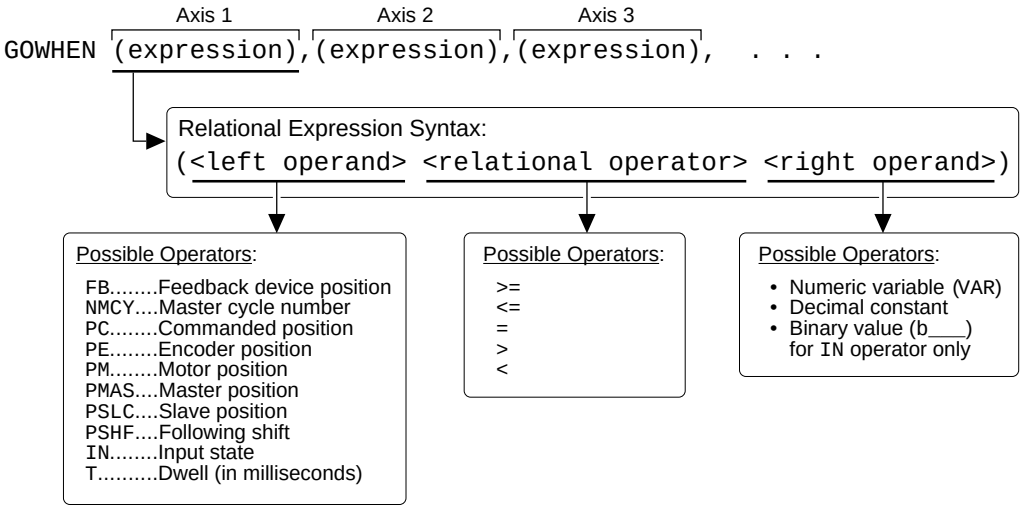
Conditional “GO”s (GOWHEN)

The GOWHEN command is used to synchronize a motion profile of an axis with a specified position count, 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.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

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 pg. 25).
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).

Further instructions about handling error conditions are provided on page 30.

ER.14 ..... Assignment & comparison operator — use in a conditional expression (see pg. 25).
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 *Trigger Functions* below.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

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 *Trigger Functions* below 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 *(Compiled Motion is discussed on page 163)*

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.

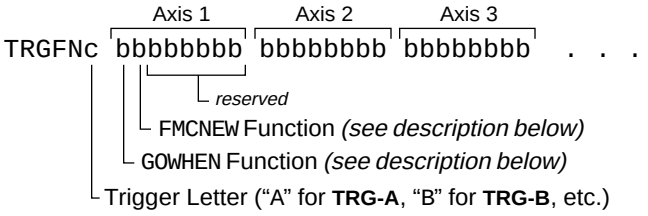
```
SCALE1              ; Enable scaling
SCLV25000,25000     ; Set velocity scaling factors
SCLD10000,10000     ; Set distance scaling factors
MC00                ; Set both axes to preset move mode
D20,20              ; Set distance end-point
COMEXC1             ; Enable continuous command execution 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.
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Trigger Functions (TRGFN)

The Trigger Functions command (TRGFN) allows you to assign additional functions to trigger inputs that have been defined as trigger interrupt inputs (INFNCi-H):

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 ("1" enables the function, "0" disables the function, "x" leaves the function bit unchanged):



GOWHEN Function: (TRGFNc1xxxxxx) 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 page 186 or to the GOWHEN command description for additional details. Axis status bit #26 (reported with TASF, TAS, or AS) is set to one (1) when there is a pending GOWHEN condition initiated by a TRGFNc1xxxxxx command; this bit is cleared when the trigger is activated or when a stop or kill command is issued.

FMCNEW Function: (TRGFNcx1xxxxx) Allows a new Following master cycle to begin when the specified trigger input (c) goes active. For additional details on master cycles, refer to page 208 or to the FMCNEW command description.

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.

Sample 6000 Code

```
INFNC26-H           ; Assign trigger B (TRG-B) on the AT6400 (input #26)
                    ; to function as a trigger interrupt input.
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.
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

CHAPTER SIX

6 Following

IN THIS CHAPTER

This chapter will help you understand Ratio Following:

- Introduction to Ratio Following 192
- Implementing Ratio Following 194
- Master Cycle Concept 207
- Technical Considerations for Following 213
- Troubleshooting for Following 223
- Following Commands (list) 225

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Ratio Following – *Introduction*

As part of its standard features, the 6000 Series Controller family allows you to solve applications requiring *Ratio Following*.

Compiled Profiles

You can pre-compile Following profiles (saves processing time). See page 166 for details.

Ratio Following is, essentially, controlled motion based on the measurement of external motion. This includes concepts such as an electronic gearbox, trackball, slave feed-to-length, as well as complex changes of ratio as a function of master position. Ratio Following can include continuous, preset, and registration-like moves in which the velocity is replaced with a ratio.

The slave may follow in either direction and change ratio while moving, with phase shifts allowed during motion at otherwise constant ratio. Ratio changes or new moves may be dependent on master position or based on receipt of a trigger input. Also, a slave axis may perform Following moves or normal time-based moves in the same application because Following can be enabled and disabled at will. *Product cycles* (operations which repeat with periodic master travel) can be easily specified with the master cycle concept (see page 207).

In Ratio Following, acceleration ramps between ratios will take place over a user-specified master distance. Product cycles can be easily specified with the master cycle concept.

This chapter highlights the capabilities of the 6000 Following features and provides application examples. If you need more details on the operation or syntax of a particular command, please refer to the ***6000 Series Software Reference***.

Before delving into the specifics of Ratio Following, read on to gain a basic understanding of how the 6000 controller *follows*.

What can be a master?

Any axis on the 6000 controller can be slaved to any transducer device input or commanded position of any other axis. This source is known as the *master input*, or *master*. Up to 4 axes (depending on which 6000 product you have) can be following at the same time with the same or different master.

The options available for master input sources differ by product (see table).

Master Input Options (✓ = yes)	AT6400	AT6200	AT6250	AT6450	610n	615n	620n	625n	6270	OEM-AT6n00
Commanded Position	✓	✓	✓	✓	n/a	n/a	✓	✓	✓	✓
Input Devices:										
• Incremental Encoder	✓	✓	✓	✓	✓	✓	✓	✓	✓	n/a
• **ANI** Analog Input **(-ANI option only)**	n/a	n/a	✓	✓	n/a	✓	n/a	✓	✓	n/a
• Linear Displacement Transducer (LDT)	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	✓	n/a

SERVOS: For servo controllers, an axis may not be slaved to its own feedback device input.

STEPPERS: For stepper controllers, an axis may be slaved to its own encoder input, as long as that axis does not need encoder feedback for encoder positioning (`ENC1` mode) or stall detection (`EPM1` mode).

For more information on assigning a master for a particular slave, refer to *Define the Master and Slave Axes* (page 194), or refer to the `FOLMAS` command description in the ***6000 Series Software Reference***.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Performance Considerations *(see also page 33)*

Position Sampling Period

Steppers – 2 milliseconds.

Servos – *system update period* (depends on SSFR and INDAX command values – see SSFR command description).

When a slave is following a master, the 6000 controller does not simply measure the master velocity to derive slave velocity. Instead, the 6000 controller samples the master position (*position sampling period*) and calculates the corresponding slave position command. This is true even if the slave is in the process of changing ratios. A slave is not simply following velocity, but rather position. With this algorithm, the master and slave position or phase relationship is maintained indefinitely, without any drift over time due to velocity measurement errors.

The 6000 controller also measures master velocity by measuring the change in master position over a number of sample periods. The present master velocity and position may be used to calculate the next commanded slave position, so the slave has no velocity-dependent phase delay. This concept is known as *Master Position Prediction* and may be enabled or disabled as needed with the FPPEN command.

GREATER DETAIL ... on the technical aspects of Following can be found on page 213, or in the associated command descriptions in the *6000 Series Software Reference*.

The 6000 controller's default Following algorithm should work well for most applications; however, you can change the Following algorithm to meet application-specific needs. For instance, suppose that the speed of the master is very slow, or has some vibration. For a case like this, the 6000 controller allows you to filter the master position signal to generate a smooth slave position command. This is known as *Master Position Filtering* and is programmed with the FFILT command.

Following Status (TFSF, TFS & FS Commands)

Many of the Following features described in this document have associated status bits that can be displayed (with the TFSF and TFS commands) or used in assignment or comparison operations (with the FS operator). The portions of this document which describe those features also summarize the related status bits.

TFS bit response format:

```
*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
```

Bit	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., slave counts counting in the opposite direction from master).
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.
* 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, minor vibration of the master will likely 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.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Implementing Ratio Following

This section covers the basic elements of implementing Ratio Following:

- Applying Following setup parameters
- Move profiles
- Performing phase shifts (FSHFC and FSHFD)
- Application scenarios:
  - Electronic gearbox
  - Trackball

Ratio Following Setup Parameters

Prior to executing a Following move, there are several setup parameters that must be specified. These parameters may be established:

☞ *Programming examples — see application examples later in this chapter.*

- Define the *master* and *slave* axes (FOLMAS)
- Define master & slave scaling factors (SCLMAS, SCLA, SCLD, and SCLV) – *if required*
- Define the slave-to-master Following ratio (FOLRN and FOLRD)
- Define the master distance (FOLMD) – *define scaling first*
- Enable the Following Mode (FOLEN)

Following Status
(see TFSF, TFS and FS commands)

Following Status bits 5-8 (see table below) are meant to 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).

Bits 7-8 represent master motion and master direction respectively. Unless the master is a commanded position of another axis, it is likely that minor vibration of the master will cause these bits 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.

Bit #	Function (yes = 1; no = Ø)
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).

Define the Master and Slave Axes (FOLMAS)

The FOLMAS command defines the masters and the slaves. The command syntax is FOLMAS<±ii>,<±ii>,<±ii>,<±ii>, where each <±ii> represents the configuration for axes 1 through 4, from left to right. *(Note that the number of axes available differs by product).* The configuration for each axis (±ii) is further defined as follows:

- Sign bit (±): Specifies the count direction of the master source which will result in positive master travel counts. The sign bit is not meant to be used simply to change the direction of slave motion. That function can be done with the sign of the D command. Rather, the sign bit is used to allow forward motion of the physical master (e.g., conveyor belt, rotating wheel, or the continuous feed of material or product) to result in positive counts. Several features described later in this document require increasing master counts for proper operation. These include Following motion in preset positioning mode (MCØ) , master cycle counting, and executing GOWHEN based on master cycle position.
- First i: Selects the master input axis number (1 through 4) that you are assigning to the slave.
- Second i: Selects the master input (input device or commanded position) according to the axis number specified with the first i. The availability of master input sources differs by product, as indicated in the table below.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

NOTE

- Servo controllers: The slave axis cannot use its own commanded position or its currently selected feedback device (encoder, ANI, or LDT) as the master input.
- Stepper controllers: The slave axis cannot use its own commanded (motor) position as the master input. Also, a slave axis that is using the encoder step mode (ENC1) cannot use its own encoder 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.

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.

As an example, suppose the **ENCODER 3** input is to be the master input for slave axis 1, and forward travel of the physical master (e.g., conveyor belt) results in negative counts on **ENCODER 3**. Given these operating constraints, you would use the FOLMAS-31,, command.

The default setting is that all axes are disabled from being slaves (FOLMAS+Ø,+Ø,+Ø,+Ø). If you do not want a particular axis to be a slave, simply leave it not configured (e.g., FOLMAS,+31,, command configures only axis #2 as a slave to encoder #3; the rest of the axes are left in the default state—not slaved). If an axis is currently configured as a slave, you can disable its slave status by putting a zero (Ø) in the parameter field (e.g., the FOLMAS,Ø,, command disables axis 2 from being a slave axis).

As soon as the master/slave configuration is specified with the FOLMAS command, a continuously updated relationship is maintained between the slave's position and the master's position. The update period for stepper controllers is 2 ms. The update period for servo controllers (*system update*) depends on the current SSFR and INDAX command values (see SSFR command description in the ***6000 Series Software Reference*** .)

FOLMAS Setting Not Saved in Non-Volatile Memory

The FOLMAS configuration is not saved in non-volatile memory. Therefore, you may wish to include it in the power-up program (STARTP) for serial-based controllers, or in the first program downloaded for bus-based controllers.

Notes for Stepper Controller Users

- If the slave axis is in encoder step mode (ENC1), then the encoder resolution (ERES) of the slave axis is also used. For that reason, *the encoder resolution may not be changed after the FOLMAS command configures an axis as a slave.*
- If the slave axis is in encoder step mode (ENC1), or if stall detect is enabled (ESTALL1), or if position maintenance is enabled (EPM1), that axis cannot use its own encoder input as the master.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Define the Master and Slave Scaling Factors (SCLMAS)

. . . . *if required*

IF SCALING IS NOT USED (SCALEØ)

Servos:

- Slave distance values are entered in the units of the currently selected feedback device (i.e., the device selected with the SFB command).

  Master distance values are entered in actual master counts (counts from the selected encoder, ANI, LDT, or commanded position).

- Velocity and accel/decel units of measure depend on the feedback source selected:

  Encoder feedback – revs/sec and revs/sec^2
  (counts output per rev depend on encoder resolution set with the ERES command).

  LDT feedback – inches/sec and inches/sec^2
  (counts output per inch depend on LDT resolution set with the LDTRES command.

  ANI feedback – volts/sec and volts/sec^2 (resolution is 819 ADC counts per volt).

Steppers:

- Slave distance values are entered in motor steps if the motor step mode is enabled (ENCØ), or encoder steps if the encoder step mode is enabled (ENC1). *If an axis is in the Encoder Step Mode (ENC1), it cannot use its own encoder input as the master.*

  Master distance values are entered in motor steps if the master is the motor position of another axis, or encoder steps of the master is an encoder.

- Velocity and accel/decel values are entered in motor revs/sec and revs/sec^2, respectively (the steps output per rev depend on the resolution setting—DRES).

Velocity and Accel/Decel Scaling

Velocity and accel/decel scaling factors for the slave are set with the SCLV and SCLA commands, respectively. *Refer to page 83 for details on using velocity and accel/decel scaling.* The SCLV and SCLA scaling factors should be set to the same value as the slave distance scaling factor (SCLD) to establish common user units (e.g., distance in inches, velocity in inches/sec, accel in inches/sec^2).

Distance Scaling

It is useful to define master and slave multipliers so later programming can take place in *user units*. The SCLD command defines the slave's distance scale factor, and the SCLMAS command defines the master's distance scale factor. The Following-related commands that are affected by SCLD and SCLMAS are listed in the table below. **NOTE**: Scaling must be enabled with the SCALE1 command before these commands will have any effect.

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

As the distance scaling factor (SCLMAS or 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 FSHFD1.99999 command would be truncated to FSHFD1.9999). 6270 users: shift the values in the "Distance Range" column one decimal place to the left.

SCLD or SCLMAS (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

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

NOTE **DEFINE SCALING FIRST** **NOTE**

If scaling is desired within a stored program, you must enable scaling (SCALE1) and define the scaling factors (SCLA, SCLV, SCLD, & 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.

NOTE **FRACTIONAL STEP TRUNCATION** **NOTE**

If you are specifying master distance values (FOLMD), when the master distance scaling factor (SCLMAS) 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 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 Distance Scaling Scenario

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

The master is a 1000-line encoder (4000 counts/rev post-quadrature) mounted to a 50 teeth/rev pulley attached to a 10 teeth/inch conveyor belt, resulting in 80 counts/tooth (4000 counts/50 teeth = 80 counts/tooth). To program in inches, you would set up the master scaling factor with the SCLMAS8ØØ command (80 counts/tooth ∗ 10 teeth/inch = 800 counts/inch).

The slave axis is a servo motor with position feedback from a 1000-line encoder (4000 counts/rev). The motor is mounted to a 4-pitch (4 revs/inch) leadscrew. Thus, to program in inches, you would set up the slave scaling factor with the SCLD16ØØØ command (4000 counts/rev ∗ 4 revs/inch = 16000 counts/inch).

Finding the Scale Factors

If the SCLD (slave) and SCLMAS (master) scale factor values are not immediately obvious, use the procedure below for help.

Slave *(applicable to servos, and steppers with encoder feedback)*:

1. Disable scaling with the SCALEØ command.
2. Servos: Issue the TFB command to display the position (based on the active feedback source).

   Steppers: Issue the TPE command to display the encoder position.

   Write down this value for later comparison.
3. Making sure the feedback device is monitoring movement of the slave, move the slave a known distance in the desired units. For instance, if you want to program in 5-inch units, move the slave 5 inches. This could be done in the jog or joystick modes, or perhaps by disabling the drive (DRIVEØ) and physically moving the slave.
4. Issue the position status command (TFB or TPE) again. The difference between the position values before and after the move, divided by the move distance in step 3, is the value you should use in the SCLD command parameter.

Master:

1. Disable scaling with the SCALEØ command.
2. Issue the TPMAS command to display the position. Write down this value for later comparison.
3. Move the master input a known distance in the desired units. For instance, if you are going to follow the motion of a conveyor belt and you want to program in 6-inch increments, move the conveyor 6 inches.
4. Issue the TPMAS command again. The difference between the TPMAS values before and after the move in step 3, divided by the distance of the move, is the value you should use in the SCLMAS command parameter.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Define the Slave-to-Master Following Ratio (FOLRN & FOLRD)

The FOLRN and FOLRD commands establish the goal ratio between the slave and master travel, just as the V command establishes the goal velocity for a typical non-Following move. The FOLRN command specifies the ratio's numerator (slave travel), and the FOLRD command specifies the ratio's denominator (master travel). If the denominator (FOLRD) is not specified, it is assumed to be 1.

FOLRNF may be used to define a final ratio for compiled Following profiles (see page 166).

FOLRN and FOLRD are specified with two positive numbers, but the resulting ratio applies to moves in both directions; the actual slave direction will depend on the direction commanded with the D command and master direction. Numeric variables (VAR) can be used with these commands for slave and/or master parameters (e.g., FOLRN(VAR1) : FOLRD3). The maximum value of the resulting quotient is 127 to 1.

For a preset Following move (MCØ mode), the FOLRN/FOLRD ratio represents the maximum allowed ratio. For a continuous move (MC1 mode), it represents the final ratio reached by the slave.

Example ☞ As an example, assume the slave-to-master ratio is set to 5-to-3 for an axis (FOLRN5 : FOLRD3). The first parameter (5) is scaled by the SCLD value to give slave steps. The second parameter (3) is scaled by the SCLMAS value to give master steps. If the SCLD setting is 25000 and the SCLMAS setting is 4000, the slave-to-master step ratio would be 5 * 25000 to 3 * 4000, or 125 slave steps for every 12 master steps.

Define the Master Distance (FOLMD)

The "master distance" for moves in the Following mode (FOLEN1) is analogous to the move time for normal time-based moves with Following disabled (FOLENØ). For time-based moves, the time required to ramp to a new velocity (MC1 mode) or move to a new position (MCØ mode) is determined indirectly by the acceleration (A), deceleration (AD), and velocity (V) command values. For Following mode moves, a ramp to a new ratio (MC1 mode) or a move to a new position (MCØ mode) takes place over a specific master distance, not over a specific time. This distance is defined directly by the user with the FOLMD command.

In other words, the FOLMD command defines the master distance over which a preset slave move will take place, or the master distance over which a continuous slave move will change from its current ratio (including zero) to the commanded ratio (ratio established by FOLRN and FOLRD).

By carefully specifying a master distance (FOLMD), a precise position relationship between master and slave during all phases of the profile is ensured.

☞ **HINT**: If the 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.

If scaling is enabled (SCALE1), the FOLMD value is scaled by the SCLMAS parameter.

Examples and more information on this topic can be found below in the section titled *Slave vs. Master Move Profiles*.

Enable the Following Mode (FOLEN1)

When an axis is configured as a slave with the FOLMAS command, it will continuously monitor the position and motion of its master, even if the slave is at rest. This allows subsequent motion to be related to the motion of the master via ratios (FOLRN/FOLRD) and ramping over master distances (FOLMD). Such moves are done with Following enabled (FOLEN1).

It is also possible, and sometimes desirable, to have the slave motion independent of master motion, yet still "aware" of master position. For example, a move may need to start at a specified master position, yet finish in a fixed time, independent of the master speed. This move would be performed with Following disabled (FOLENØ).

Following may be enabled or disabled between moves, as needed, without affecting the monitoring of the master.

If a move is performed with Following disabled, its motion profile is determined by the acceleration, deceleration, and velocity specified with the A, AD, and V commands. Its motion is the same as if the axis were not configured as a slave, but the axis does monitor the master.

If a move is performed with Following enabled, its profile is determined by the specified master distance (FOLMD) and Following ratio (FOLRN/FOLRD). The next section describes such profiles.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Slave vs. Master Move Profiles

Following Status (TFSF, TFS, and FS) bits 1-4 indicate the status of slave following motion. They mimic the meaning and organization of Axis Status (TASF and AS) bits 1-4, except that each bit indicates the current state of the ratio, rather than the current state of the velocity:

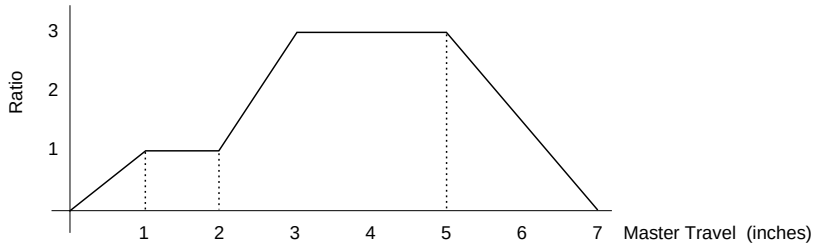
Bit #	Function (YES = 1; NO = Ø)
1	Slave in Ratio Move ..... A Following move is in progress
2	Ratio is Negative ..... The current ratio is negative (i.e., the slave counts are counting in the opposite direction from the master counts).
3	Slave Ratio Changing..... The slave is ramping from one ratio to another (including a ramp to or from zero ratio).
4	Slave At Ratio ..... The slave is at constant non-zero ratio.

Continuous Positioning Mode Moves

For Following moves in the continuous positioning mode (MC1), FOLMD specifies the exact master travel distance over which the slave ratio changes. This will be required for any application that uses multiple ratios and continuous moves for the construction of precisely defined multi-segment moves.

☞ **HINT**: If the 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.

In the profile below, the first two moves each change ratio over one master inch, and the final ramp to zero takes place over two master inches.



Example ☞ In the sample 6000 code below, assume the slave has a 1000-line encoder on axis 1, connected to a 2-pitch leadscrew. This gives 8000 slaves steps per inch. The master is a toothed belt with a pulley connected to encoder 2, such that there are 800 master steps per inch.

```
COMEXC1             ; Allow commands during motion
FOLMAS21            ; Define axis 1 master to be encoder input #2
FOLEN1              ; Enable Following on axis 1 (will follow encoder #2)
SCALE1              ; Enable scaling
SCLD8000            ; Set axis 1 scaling so that slave commands are in inches
SCLMAS800           ; Set axis 1 scaling so that master commands are in inches
FOLMD1              ; Slave to change ratio over 1 inch of the master travel
FOLRD1              ; Set Following ratio denominator to 1 for subsequent ratios
D+                  ; Set direction positive
MC1                 ; Mode set to continuous clockwise moves
FOLRN1              ; Set Following ratio numerator to 1 (ratio set to 1:1)
FMCNEW1             ; Restart master cycle counting
GO1                 ; Start axis 1 from rest to reach velocity of master
                    ; (encoder #2)
WAIT(1FS.4=B1)      ; Wait until slave is at ratio
FOLRN3              ; Set Following ratio numerator to 3 (ratio set to 3:1)
GOWHEN (1PMAS>=2)   ; Enable motion pre-processing so that the next ramp
                    ; begins at master position 2
GO1                 ; Slave starts ratio change to 3 to 1 at master position 2
FOLRN0              ; Set Following ratio numerator to zero (ratio is 0 to 1)
FOLMD2              ; Have slave change ratio over 2 inches of master travel
WAIT(1AS.26=b0)     ; Wait until the previous ramp is started
                    ; (GOWHEN bit in axis status register is cleared)
WAIT(1FS.3=b0)      ; Wait until the previous ramp is finished
GOWHEN(1PMAS>=5)    ; Enable motion pre-processing so that slave motion
                    ; begins at master position 5
GO1                 ; Wait for master to reach 5 revolutions before
                    ; slave starts ratio change to 0 (zero)
```

This command is for steppers only →

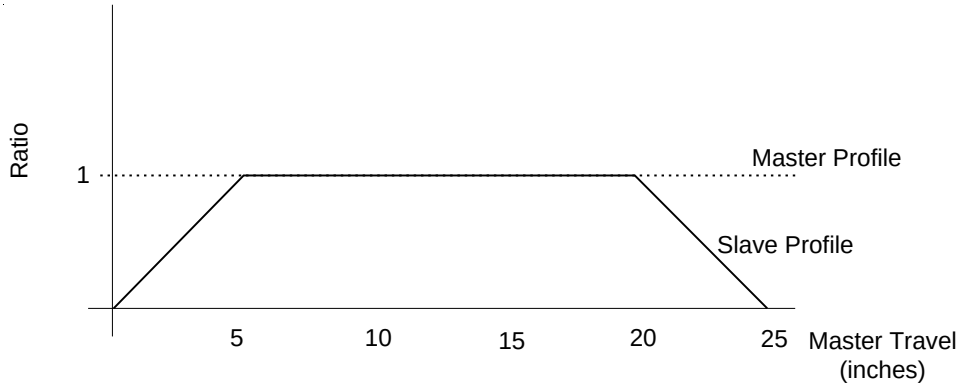
Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Preset Positioning Mode Moves

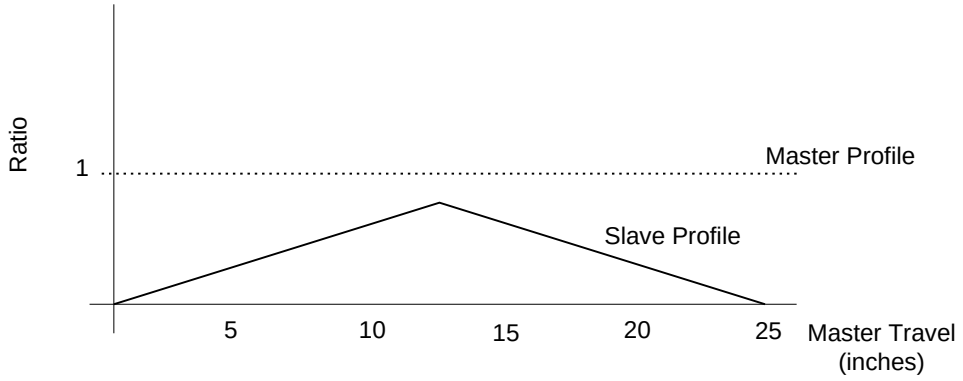
For preset positioning mode (MCØ) moves, the FOLMD parameter is the master distance over which the entire slave move is to take place.

As an example, a slave is to move 20 inches over a master distance of 25 inches with a maximum ratio of 1:1 (ratio set with the FOLRN1 and FOLRD1 commands). The program and a diagram of the move profiles are provided below.

```
FOLMAS31          ; Define axes 1 master to be encoder input port 3
FOLMD25           ; Define slave to perform the move over 25 inches
                  ; of the master (encoder 3)
MC0               ; Set positioning mode to preset
MA0               ; Set preset positioning mode to incremental
D20               ; Set slave distance to 20 inches
FOLRN1            ; Set Following ratio numerator to 1
FOLRD1            ; Set Following ratio denominator to 1 (ratio is 1:1)
GO1               ; Perform the slave move
```



If the master distance specified is too large for the slave distance and ratio (FOLRN and FOLRD) commanded, the slave will never actually reach the commanded ratio, and the move profile will look similar to that below. Here, the FOLMD is 25 inches and the slave is commanded to move 10 inches:



If the master distance is too small for the slave distance and ratio commanded, the 6000 controller will not perform the move at all. For example, if the FOLMD is 25, the ratio is 1:1, and the slave is commanded to move 30 inches, the move will not even be attempted. The error message "INVALID DATA" will be displayed (depending on the ERRLVL setting) and program execution will continue.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Performing Phase Shifts

Following Status (TFSF, TFS and FS) 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.

Bit #	Function (YES = 1; NO = Ø)
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.

When a slave is following a master continuously, it may be necessary to adjust, or *shift*, the Following *phase* (slave's position with respect to the master) independent of motion due to ratio moves. The FSHFC and FSHFD commands allow time-based slave moves to be superimposed upon ratio Following moves. Because phase shifts are time-based, they are independent of master motion; in fact, the master may be at rest and a shift may still be performed.

Use the FSHFD command to perform a preset shift move with a specific change in slave phase. The FSHFD distance value will be scaled by SCLD if scaling is enabled (SCALE1).

Use the FSHFC command to superimpose a continuous shift move in the positive (FSHFC1) or negative (FSHFC2) direction. The FSHFC parameters stop (Ø) and kill (3) can be used to halt a continuous FSHFC move or a preset FSHFD move without affect the ratio motion.

The most recently defined velocity and acceleration (i.e., the V and A values) for the slave will determine the basis for the superimposed shift move profile for both FSHFC and FSHFD moves. The commanded velocity of the FSHFC or FSHFD move will be added to the current velocity at which the slave is performing the Following move. For example, assume a slave is traveling at 1 rps in the positive direction as a result of 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.

For servos, shifting may be performed whenever Following is enabled (FOLEN1). For steppers, this may only be done while Following is enabled and the slave is either not in a move, or is in continuous positioning mode (MC1) and moving at constant ratio. For both products, TFS/FS bit 9 indicates when a shift is allowed.

The current slave position (TPSLV value) and the net slave shift accumulated since the most recent FOLEN1 command (TPSHF value) may be read into numeric variables (VAR) using the PSLV and PSHF commands, respectively (e.g., VAR6=2PSLV). They may also be used for subsequent decision making (e.g., IF(3PSHF<6), GOWHEN(1PSLV>VAR2), etc.).

The TPSHF and PSHF values are set to zero each time the Following mode is enabled (FOLEN1), even if the slave is already in Following mode. This provides a way of clearing these values for programming convenience.

Note that the distance traveled during the time-based deceleration due to stop, kill, or limits is included in the PSHF value. By comparing "before and after" values of PSHF, a 6000 program may calculate how much shift was required to perform visual- or sensor-based alignment of a master/slave phase relationship.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Phase Shift Examples

An FSHFC or 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. Below are programming examples to demonstrate both shift methods.

FSHFC Example

An operator is visually inspecting the slave's continuous Following 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 by pressing a pushbutton switch) that will allow the operator to move the slave at a superimposed correction speed until the operator chooses to have the slave start tracking the master again (by releasing the pushbutton). The programming example below illustrates this.

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 until the button is released. After the adjustment, the program continues on as before.

```
DEL SHIFT          ; Delete program before defining
DEF SHIFT          ; Begin definition of program called SHIFT
COMEXS1            ; Continue command execution after stop
COMEXC1            ; Continue command execution during motion
FOLMAS21           ; Axis 2 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
A25                ; Set acceleration
AD18               ; Set deceleration
V5                 ; Set velocity
D+                 ; Set direction to positive
MC1                ; Select continuous positioning mode
GO1                ; Start following master continuously
VARB1=b10          ; Define input pattern #1 and assign to VARB
VARB2=b01          ; Define input pattern #2 and assign to VARB
$TESTIN            ; Define label called TESTIN
IF(IN=VARB1)       ; IF statement (if 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=B0)      ; Continue shift until input bit #1 is deactivated
FSHFC0             ; Stop shift move
WAIT(1FS.10=B0)    ; Wait until the shift is completed (steppers only)
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=B0)      ; Continue shift until bit #2 is deactivated
FSHFC0             ; Stop shift move
WAIT(1FS.10=B0)    ; Wait until the shift is completed (steppers only)
JUMP TESTIN        ; Return to main program loop
END                ; End definition of program called SHIFT
```

This line is for steppers only →

This line is for steppers only →

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

FSHFD Example

In this example, the slave follows a master that moves in a continuous cycle. Once each cycle, the master and slave both pick parts. The master's part is detected by a sensor connector to trigger B, and the slave's part is detected by a sensor connected to trigger A. After both parts are detected, they must be aligned. The sensors are mounted 2 inches apart from each other, so that proper alignment would result in 2 inches of slave travel between detection of the master's part and detection of the slave's part.

The slave position is sampled when each of the sensors activates. The difference between the slave positions is compared to the required 2 inches. If the measured difference is greater than or less than 2 inches, then a shift move to correct the alignment is made. At that point, the slave will then start tracking the master again. The slave (axis #1) is continually following the master at a 1:1 ratio.

```
DEL ALIGN           ; Delete program before defining
DEF ALIGN           ; Begin definition of program called ALIGN
COMEXC1             ; Allow continuous command execution during motion
INFEN1              ; Enable input functions
FOLMAS31            ; Axis 3 Encoder input is the master for slave axis 1
FOLRN1              ; Set slave-to-master ratio numerator to 1
FOLRD1              ; Set slave-to-master ratio denominator to 1
                    ; (ratio set to 1:1)
FOLEN1              ; Enable Following mode on axis #1
MC1                 ; Enable continuous positioning mode
D+                  ; Set direction to positive
GO1                 ; Start Following master continually
INFNC25-H           ; Enable trigger input A to latch position of slave
                    ; when the slave's part is detected
INFNC26-H           ; Enable trigger input B to latch position of slave
                    ; when the master's part is detected
$SYNCLP             ; Main loop where synchronizing moves occur
WAIT(SS.25=b1 AND SS.26=b1) ; Wait for both slave and master inputs to occur
VAR10=1PMCA         ; Load VAR10 with the slave motor position due to
                    ; slave input activation
VAR11=1PMCB         ; Load VAR11 with the slave motor position due to
                    ; master input activation
VAR10=1PCCA         ; Load VAR10 with the slave commanded position due
                    ; to slave input activation
VAR11=1PCCB         ; Load VAR11 with the slave commanded position due
                    ; to master input activation
VAR12=VAR10-VAR11   ; Load VAR12 with the offset distance
VAR13=VAR12-2       ; Calculate the required shift
FSHFD(VAR13)        ; Perform synchronization move of distance in VAR13
JUMP SYNCLP         ; Return (jump) to main program loop
END                 ; End of program
```

This line is for steppers only →

This line is for steppers only →

This line is for servos only →

This line is for servos only →

Summary of Ratio Following Commands

FOLEN .....Enables or disables Following mode

FOLMAS.....Defines masters for slave axes

FOLMD .....Defines the master distance over which slave acceleration or moves are to take place

FOLRN and FOLRD.....Establishes the maximum slave-to-master ratio for a preset move or the final ratio for a continuous move (FOLRN for the numerator and FOLRD for the denominator)

FSHFD .....Initiates preset advance or retard (shift) of slave position during continuous Following moves

FSHFC .....Initiates continuous advance or retard (shift) of slave position (or kills or stops the shift portion of motion) during continuous Following moves

SCLD.....Sets the slave distance scale factor

SCLMAS.....Sets the master distance scale factor

TPSHF or [PSHF] .....Transfers or assigns the net position shift since constant ratio

TPSLV or [PSLV] .....Transfers or assigns the current slave position

TVMAS or [VMAS] .....Transfers or assigns the velocity of the master axis

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Electronic Gearbox Application for Ratio Following

An electronic gearbox is a classic application for Ratio Following. Suppose we need a three-output gearbox, with all three outputs geared off the same input. Also, each gear ratio must be individually programmed. In this example, a 1000-line encoder is mounted to the input shaft of a master motor, giving 4000 master counts per revolution after quadrature. This encoder is fed into the encoder input on axis #4 (**ENCODER 4** connector) of the 6000 controller. The motors on axes 1, 2, and 3 have resolutions of 2000, 4000, and 5000 steps/revolution.

In this example, a precise position relationship is not required between master and slaves, but a ratio change during motion is required. The ratio change will take place over one master revolution in order to avoid abrupt acceleration of the slave. The slaves will accelerate to their initial ratios (in terms of revolutions), and after 10 seconds the gear ratio on each axis will change to their final ratios.

In this example, **ENCODER 4** is specified as the master. This means that this *external* master encoder is wired to the 6000 controller's axis #4 encoder input.

Program
(code portion)

```
SCALE1              ; Enable scaling
SCLD2000,4000,5000  ; Set slave scale factors equal to the motor resolutions
SCLA2000,4000,5000  ; Set acceleration scale factors equal to the
                    ; motor resolutions
SCLMAS4000,4000,4000  ; Master scale factor to number of pulses per rev
COMEXC1             ; Allow continuous command execution during motion
FOLMAS+41,+41,+41   ; Encoder #4 is master axis for slave axes #1 - #3.
                    ; The + sign indicates that the master input is not
                    ; inverted before it is read as master counts.
FOLEN111            ; Enable slaves to follow
FOLMD1              ; Change to new ratio over one master revolution
FOLRN1,3,2          ; Set slave-to-master ratio numerators to 1, 3, and 2
FOLRD1,1,1          ; Set all slave-to-master ratio denominators to 1
                    ; (initial Following ratio for axis 1 is 1:1, axis 2
                    ; is 3:1, and axis 3 is 2:1)
MC111               ; Enable continuous mode
GO111               ; Begin slave continuous Following move
TIMST0              ; Reset and start the timer
FOLRN10,6,1         ; Set slave-to-master ratio numerators to 10, 6, and 1
FOLRD1,7,2          ; Set slave-to-master ratio denominators to 1, 7 and 2
                    ; (change Following ratios: axis 1 is 10:1, axis 2 is 6:7,
                    ; axis 3 is 1:2)
WAIT(TIM>=10000)    ; Wait 10 seconds to change to new ratio
GO111               ; Start moving to new ratio
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Trackball Application for Ratio Following Motion

A trackball is a two-axis, two-dimensional positioning device; just as a mouse is used to position the cursor on a computer screen, a trackball could be used to position an X-Y stage.

In this example, a two-axis trackball is needed which can do fine and coarse positioning of an X-Y stage. The fine or coarse setting is selected by the user with a two-position switch connected to programmable input #1 on the 6000 controller. Programmable input #2 on the 6000 controller is used to switch back and forth from trackball to standard point-to-point positioning mode. *Unlocking* the stage from the trackball is necessary because of other point-to-point move requirements elsewhere in the 6000 controller program.

The trackball housing has two encoders mounted at 90 degrees to each other which are driven by rubber wheels in contact with the ball. The stage is driven by motors and leadscrews.

For one inch of trackball motion to result in one inch of stage motion, the slave-to-master ratio must be 10-to-1; this will be the ratio for coarse positioning. The fine positioning ratio will be one tenth of that, or 1-to-1. When programmable input #1 is low, coarse positioning is selected, and when programmable input #2 goes low, the stage becomes locked to the trackball. Each change of state of inputs #1 and #2 calls a different subroutine in the 6000 controller program; however, the ratios can only change if the stage is locked to the trackball positioning mode.

The trackball is initially unlocked and fine positioning is selected.

Program

```
SCALE1              ; Enable & define scale factors prior to loading program
SCLD4000,4000       ; Slave axes 1 and 2 have 4000 counts per rev
                    ; resolution post-quadrature
SCLV4000,4000       ; Set velocity scaling factors
SCLA4000,4000       ; Set acceleration scaling factors
SCLMAS200,200       ; Master axis 3 and 4 have 200 counts per rev
                    ; resolution post-quadrature

DEL UNLOCK          ; Delete program before defining
DEF UNLOCK          ; Program that unlocks the stage from the trackball
S11                 ; Stop moves
WAIT(1AS.1=B0 AND 2AS.1=B0)  ; Wait for motion to stop on both axes
ONIN.2-1            ; Set up input 2 to lock trackball to stage
FOLEN00             ; Stop Following mode
ONP LOCK            ; Select LOCK as an ON program
JUMP WAITLP         ; Return to main loop
END                 ; End program definition

DEL LOCK
DEF LOCK            ; Program that locks the stage to the trackball
FOLEN11             ; Enable Following on both axis
IF(VAR1=0)          ; If in the FINE mode, set ratio to 0.5:1
FOLRN.5,.5
ELSE
FOLRN1.5,1.5        ; If in the COARSE mode, set ratio to 1.5:1
NIF
ONIN.2-0            ; Set up input 2 to unlock trackball from stage
GO11                ; Start following the trackball
ONP UNLOCK          ; Select UNLOCK as ON program
JUMP WAITLP         ; Return to main loop
END                 ; End program definition

DEL COARSE
DEF COARSE          ; Declare COARSE label
FOLRN1.5,1.5        ; Coarse positioning ratio of 1.5 to 1
GO11                ; Move to begin travel at new ratio
VAR1=1              ; Flag to indicate we are in coarse mode
END                 ; Return to main loop
```

This line is for steppers only →

(Continued)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Trackball Program *(Continued)*

```
DEL FINE
DEF FINE            ; Subroutine to assign fine positioning
FOLRN.5,.5          ; Fine positioning ratio is 0.5:1
GO11                ; Move to begin travel at new ratio
VAR1=0              ; Flag to indicate we are in fine mode
END                 ; Return to main loop

DEL TRACK
DEF TRACK                ; Main track ball program
IF(IN.1=b1 AND VAR1=0)   ; If input 1 is set to 1 and we are
                         ; currently in fine mode, then enter coarse mode
  GOSUB COARSE           ; Set high ratio
NIF
IF(IN.1=b0 AND VAR1=1)   ; If input 1 is set to 0 and we are
                         ; currently in coarse mode, then enter fine mode
  GOSUB FINE             ; Set low ratio
NIF
IF(LIM.1=b0 OR LIM.2=b0) ; If a limit is hit, allow track ball to move off
D~                  ; Back off of the limit, axis 1
GO1
NIF
IF(LIM.4=b0 OR LIM.5=b0)
D,~                 ; Back off of the limit, axis 2
GOX1
NIF
END

DEL MAIN
DEF MAIN            ; Begin definition of main program
V1,1                ; Set non-Following move parameters
A99,99
LH3,3,0,0
FOLMAS+31,+41       ; Encoder #3 is master axis for slave axes #1 and
                    ; Encoder #4 is master axis for slave axes #2.
                    ; The slave axes will move in the same direction as
                    ; the master.
FOLRN.5,.5
INDEB25,250         ; Noisy switch debounce of 250 milliseconds
INDEB26,250
FOLRD1,1            ; Initial slave-to-master ratio is set to fine
                    ; positioning (0.5:1)
VAR1=0              ; Flag set to fine positioning
MC11                ; Set both axes 1 and 2 to continuous positioning mode
FOLEN00             ; Following is initially disabled
COMEXC1             ; Continue command execution during motion.
COMEXS1             ; Continue command execution after stop
COMEXL11            ; Continue command execution after a limit is hit
INFEN1              ; Enable input functions
SGP20,20            ; Set servo gains
SGV5,5
DRIVE1100           ; Enable drives
DRFLVL11XX          ; Set drive fault level for Compumotor 670-T drives
ONP LOCK            ; Select LOCK as an ON program
ONIN.26-1           ; Trigger B locks trackball to stage
ONCOND1000          ; Inputs enabled for interrupts
$WAITLP             ; Main program loop
IF(IN.26=b1)        ; If trigger input B is set to 1 (stage locked),
                    ; enter trackball mode
  GOSUB TRACK
NIF                 ; End of IF statement

; ************************************************************
; *  Other user programs can be added here for performing     *
; *  motion when the stage is not locked to the trackball.    *
; ************************************************************

JUMP WAITLP         ; Return to main loop
END                 ; End program definition
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Master Cycle Concept

Ratio Following can also address applications that require precise programming synchronization between moves and I/O control based on master positions or external conditions. The concept of the master cycle greatly simplifies the required synchronization.

A master cycle is simply an amount of master travel over which one or more related slave events take place. The distance traveled by the master in a master cycle is called the *master cycle length*. A *master cycle position* is the master position relative to the start of the current master cycle. The value of master cycle position increases as positive-direction *master cycle counts* are received, until it reaches the value specified for master cycle length. At that point, the master cycle position becomes zero, and the *master cycle number* is incremented by one—this condition is called *rollover*.

The master cycle concept is analogous to minutes and hours on a clock. If the master cycle is considered an hour, then the master cycle length is 60 minutes. The number of minutes past the hour is the master cycle position, and current hour is the master cycle number. In this analogy, the master cycle position decrements from 59 to zero as the hour increases by one.

By specifying a master cycle length, periodic actions may be programmed in a loop or with subroutines which refer to cycle positions, even though the master may be running continuously. To accommodate applications where the feed of the product is random, the start of the master cycle may be defined with trigger inputs. Two types of *waits* are also programmable to allow suspension of program operation or slave moves based on master positions or external conditions.

Master Cycle Commands

Following Status (TFSF, TFS and FS) 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:

Bit #	Function (YES = 1; NO = Ø)
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.

Master Cycle Length (FMCLEN)

The FMCLEN command is used to define the length of the master cycle. The value entered with this command is scaled by the SCLMAS parameter to allow specification of the master cycle length in user units. This parameter must be defined before those commands which wait for periodically repeating master positions are executed.

The default value of FMCLEN is zero, which means the master cycle length is practically infinite (i.e., 4,294,967,246 steps, after scaling). If a value of zero is chosen, the master cycle position will keep increasing until this very high value is exceeded or a new cycle is defined with the FMCNEW command (or triggered after a TRGFNCx1 command) described below. If a non-zero value for FMCLEN is chosen, the internally maintained master cycle position will keep increasing until it reaches the value of FMCLEN. At this point, it immediately rolls over to zero and continues to count.

The master cycle length may be changed with the FMCLEN command even after a master cycle has been started. The new master cycle length takes affect as soon as it is issued. If the new master cycle length is greater than the current master cycle position, the cycle position will not change, but will rollover when the new master cycle length is reached. If the new master

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

cycle length is less than the current master cycle position, the new master cycle position becomes equal to the old cycle position minus one or more multiples of the new cycle length.

Example Code

```
FMCLEN23,10,12,22   ; Set master cycle length for all four axes:
                    ; (axis 1: 23 units; axis 2: 10 units;
                    ; axis 3: 12 units; and axis 4: 22 units)
```

Restart Master Cycle Counting (FMCNEW or TRGFNcx1xxxxxx)

Once the length of the master cycle has been specified with the FMCLEN command, master cycle counting may be restarted immediately with the FMCNEW command, or based on activating a trigger input as specified with the TRGFNcx1xxxxxx command. The new master cycle count is started at an initial position specified with the FMCP command (see below).

When the TRGFNcx1 command is used, the restart of master cycle counting is pending activation of the specified trigger. If an FMCNEW command is issued while waiting for the specified trigger to activate, counting is restarted immediately with the FMCNEW command, and the TRGFNcx1xxxxxx command is canceled.

When using TRGFNcx1xxxxxx

- Before the TRGFN command can be used, you must first assign the trigger interrupt function to the specified trigger input with the INFNCi-H command, where "i" is the input number of the trigger input desired for the function (input bit assignments vary by product).
- Because the 6000 controller program will not wait for the trigger to occur before continuing on with normal program execution, a WAIT or GOWHEN condition based on PMAS will not evaluate true if the restart of master cycle counting is pending the activation of a trigger. To halt program operation, the WAIT command can be used.

A new master cycle will restart automatically when the total master cycle length (FMCLEN value) is reached. This is useful in continuous feed applications.

Example Code

```
FMCNEW11xx          ; Restart new master cycle counting on axes 1 and 2
INFNC26-H           ; Assign input #26 (trigger B) the trigger interrupt
                    ; function (prerequisite to using the TRGFN features)
TRGFNBx1xxxxxx x1   ; When trigger input B (TRG-B) goes active,
                    ; restart new master cycle counting on axes 1 and 2
```

Initial Master Cycle Position (FMCP)

The FMCP command allows you to assign **for the first cycle only**, an initial master cycle position to be a value other than zero. When master cycle counting is restarted with the FMCNEW command or with the trigger specified in the TRGFNcx1 command, the master cycle position takes the initial value previously specified with the FMCP command. The value for FMCP is scaled by SCLMAS if scaling is enabled (SCALE1)

FMCP was designed to accommodate situations in which the trigger that restarts master cycle counting occurs either before the desired cycle start, or somewhere in the middle of what is to be the first cycle. In the former case, the FMCP value must be negative. The master cycle position is initialized with that value, and will increase right through zero until it reaches the master cycle length (FMCLEN). At that point, it will roll over to zero as usual.

The continuous cut-to-length example below illustrates the use of a negative FMCP (a trigger that senses the motion of the master is physically offset from the master position at which some action must take place). If it is desired that the first cycle is defined as already partially complete when master cycle counting is restarted, the FMCP value must be greater than zero, but less than the master cycle length.

To give a value for FMCP which is greater than master cycle length is meaningless since master cycle positions are always less than the master cycle length. The 6000 controller responds to this case as soon as a new master cycle counting begins by using zero instead of the initial value specified with FMCP.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Transfer and Assignment/ Comparison of Master Cycle Position and Number

The current master cycle position and the current master cycle number may be displayed with the TPMAS and TNMCY commands, respectively. These values may also be read into numeric variables (VAR) at any time using the PMAS and NMCY commands (e.g., VAR6=NMCY).

Very often, the master cycle number will be directly related to the quantity of product produced in a manufacturing run, and the master cycle position can be used to determine what portion of a current cycle is complete.

Position Sampling Period

Steppers – 2 millisec.

Servos – *system update period* (depends on SSFR & INDAX command values – see SSFR command).

The master cycle number is sampled once per *position sampling period* (see note, left). If the master cycle length (FMCLEN) divided by the master's velocity (VMAS) is less than the position sampling period, then the sample (TNMCY or NMCY value) may not be accurate.

Details on using PMAS in conditional expressions is provided below in *Using Conditional Statements with PMAS*.

Using Conditional Statements with Master Cycle Position (PMAS)

The current master cycle position (PMAS) value may be used in comparison expressions, just like other position variables such as PC, PM, PE, and FB. PMAS is a special case, however, because its value rolls over to zero when the master cycle length (FMCLEN) is met or exceeded. This means that PMAS values greater than or equal to the master cycle length will never be reported with the TPMAS command, or with expressions such as (VAR1=1PMAS).

The other fact that makes PMAS special is that master cycle counting may be restarted after the command containing the PMAS expression has been executed. Either the FMCNEW command or the TRGFNcx1 command may be used to restart counting, each with a different effect on the evaluation of PMAS.

The treatment of PMAS in comparison expressions depends on the command using the expression, as described below. WAIT and GOWHEN are treated as special cases.

IF, UNTIL, and WHILE

These commands evaluate the current value of PMAS in the same way that TPMAS does (i.e., PMAS values will never be greater than or equal to the master cycle length). With these commands, avoid comparing PMAS to be greater than or equal to variables or constants which are nearly equal to the master cycle length, because rollover may occur before a PMAS sample is read which makes the comparison true. If such a comparison is necessary, it should be combined (using OR) with a comparison for master cycle number (NMCY) being greater than the current master cycle number.

Also, master cycle counting restart may be pending activation of a trigger, but this will not affect the evaluation of PMAS for IF, WAIT, and WHILE. It is simply evaluated based on counting currently underway.

WAIT and GOWHEN

These commands evaluate the current value of PMAS differently than TPMAS does, in such a way that it is possible to compare PMAS to variables or constants which are greater than or equal to the master cycle length and still have the comparison be reliably detected.

Effectively, PMAS is evaluated as if the master cycle length were suddenly set to its maximum value (2^{32}) at the time the WAIT or GOWHEN command is encountered. It eliminates the need to OR the PMAS comparison with a comparison for master cycle number (NMCY) being greater than the current master cycle number. Such multiple expressions are not allowed in the GOWHEN command, so this alternative evaluation of PMAS offers the required flexibility.

This method of evaluation of PMAS allows commands which sequence slave events through a master cycle to be placed in a loop. The WAIT or GOWHEN command at the top of the loop can execute, even though the actual master travel has not finished the previous cycle. If it is desired to WAIT or GOWHEN for a master cycle position of the next master cycle, the variable or constant specified in the command should be calculated by adding one master cycle length to the desired master cycle position.

Finally, master cycle counting restart may be pending activation of a trigger (TRGFNcx1), and this will suspend the evaluation of PMAS for these commands. PMAS is not sampled, and the comparison evaluates as false. During this time, if the pending status of master cycle counting restart is aborted with FMCNEWØ, the GOWHEN condition is also cleared, and any motion profile of any axis waiting on *that* PMAS comparison will be canceled. Otherwise, when master cycle counting is restarted by a trigger, evaluation takes place as described above. This allows GOWHEN to include waiting on a trigger without explicitly including it in the GOWHEN expression.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Synchronizing Following Moves with Master Positions

A final special case allows perfect synchronization between the start of a Following motion profile of a slave axis and a specified position of its master. If a GOWHEN(nPMAS >= xxx) expression is used to synchronize a slave with its own master, with the operator specifically ">=", a special synchronization occurs. Although it may be impossible for the 6000 product to sample the exact master position specified, the Following motion profile is calculated from master travel based on that position. This allows for the construction of profiles in which the synchronization of master and slave positions is well defined and precisely maintained. This feature requires positive travel of the master, which can be achieved with the appropriate sign for the FOLMAS specification.

Summary of Master Cycle and Wait Commands

FMCLEN ..... Defines the length of the master cycle

FMCNEW ..... Immediately restart master cycle counting

FMCP ..... Defines the initial position of a new master cycle

GOWHEN() ..... GOWHEN() suspends execution of the next move on the specified axis or axes until the specified conditional statement (based on T, IN, FB, NMCY, PC, PE, PM, PMAS, PSLV, or PSHF) is true

TNMCY or [NMCY] ..... Transfers or assigns the current master cycle number

TPMAS or [PMAS] ..... Transfers or assigns the current master cycle position

TRGFN ..... TRGFNc1xxxxxxx initiates a GOWHEN, suspending execution of the next slave move until the specified trigger input (c) goes active. TRGFNcx1xxxxxx causes master cycle counting to restart when the specified trigger input (c) goes active.

WAIT() ..... WAIT() suspends program execution until the specified conditional statement (based on PMAS, FS, NMCY, PSHF, PSLV, or VMAS) is true;
WAIT(SS.i=b1) suspends program execution until a trigger input is activated. The "i" is the programmable input bit number corresponding to the trigger input. (Input bit assignments vary by product; refer to the Programmable I/O Bit Patterns table on page 107 to determine the correct bit pattern for your product.)

AS and TAS bit #26..... AS and TAS (and TASF) bit #26 is set when there is a profile suspended pending GOWHEN condition, initiated either by a GOWHEN command or a TRGFNc1xxxxxxx command; this bit is cleared when the GOWHEN condition is true or when a stop or kill command is issued.

ER and TER bit #14..... ER, TER, and TERF bit #14 is set if the GOWHEN condition is already true when the GO, GOL, FSHFC, or FSHFD command is given (ERROR bit #14 must first be enabled to check for this condition)

NOTE

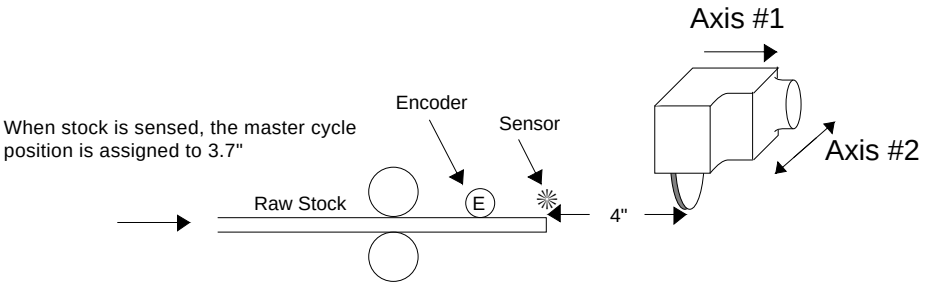
The continuous cut-to-length application example below illustrates the use of the master cycle concept and the commands above.

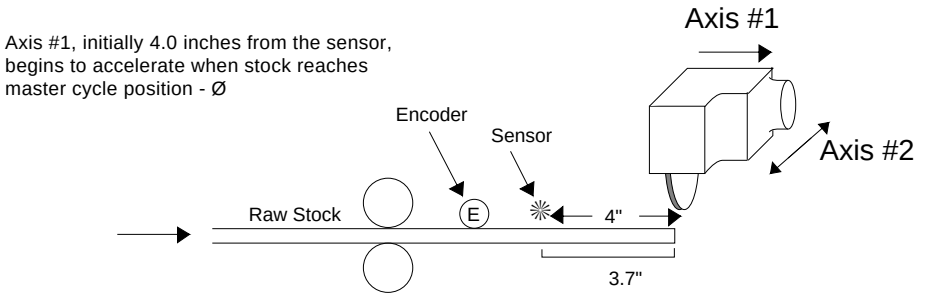
Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

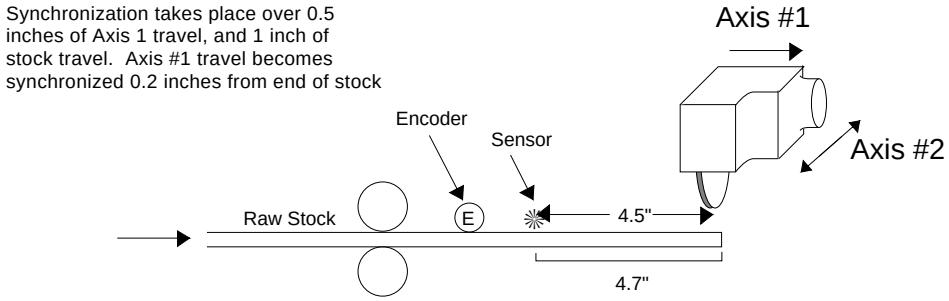
Continuous Cut-to-Length Application

This application requires automobile trim to be cut to a pre-defined length. The saw is controlled by axes #1 and #2 on the 6000 controller. It must be moving with the material while the cut is being made (axis #1), and also move perpendicular to the trim (axis #2) to actually make the cut. The trim comes in long stock which moves continuously under the cutting area.

The leading edge of the trim stock is detected with a sensor connected to trigger #1 which is located 4 inches from the home position of the saw. Axis #1 will be following the trim based on an encoder mounted on the trim via a friction wheel. The encoder is a 1000-line encoder and the wheel is geared to give 2 revolutions per inch of trim, resulting in 8,000 post-quadrature steps per inch of trim. Axis #1 has a resolution of 4,000 steps per rev and is connected to a 2-pitch leadscrew 24 inches in length. Axis #2 is similar in mechanics but its length is 10 inches. The travel on Axis #1 will be controlled by the speed at which axis #2 makes its cut. The travel on axis #2 is a fixed speed of 5 inches per second, with a fixed cross stroke of 5 inches. Limit switches are in place for safety.







Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The master cycle length will be set equal to the desired cut length (36" in the example below), which the operator can change by modifying variable VAR1. The cut cycle will be a continuous loop, but the first cut will be made 0.2 inches from the end of the stock to ensure an even first edge. Axis #1 will accelerate to the desired tracking ratio over 1 inch of master travel for all cuts. Assume that the home position of both axes is at position 0 inches.

The Cut-to-Length example takes advantage of being able to change master cycle length, while being careful to change it only at the beginning of a current cycle. This ensures that the current master cycle position will be less than the new master cycle length, and will not change as a result of a change in cycle length. In this example, the master cycle length and corresponding waits are redefined every cycle to the current value of VAR1. The value of VAR1 becomes the cut length, and can be changed via remote command during program execution. With minor modifications, the cut lengths and the number of iterations could be read from DATA commands (e.g., in a teach mode application).

Program

```
SCALE1              ; Enable scaling
SCLD8000,8000       ; Set axes #1 & #2 scale factors for programming
                    ; in inches
SCLV,8000           ; Axis #2 velocity scale factor for inches/sec
SCLA,8000           ; Axis #2 accel scale factor for inches/sec/sec
SCLMAS8000          ; Master scale factor for programming in inches
DEF CUTLEN          ; Start definition of Cut-to-Length program
COMEXC1             ; Enable continuous command execution mode
COMEXS1             ; Continue execution if stop is issued
INFEN1              ; Enable input functions
INFNC25-H           ; Enable trigger input A for TRGFN use
OUTFEN1             ; Enable output functions
A20,20              ; Set acceleration
V5,5                ; Set velocity
MA11                ; Absolute positioning mode for non-Following moves
VAR1=36             ; Desired cut length is 36"
VAR2=4              ; Sensor is 4" from home position of axis #1
VAR2=VAR2+0.2       ; 1st cut to be 0.2" from end of stock
VAR3=1              ; FOLMD to be set to 1"
VAR4=VAR3/2         ; Slave will travel 1/2" when accelerating to
                    ; 1:1 ratio while master travels 1"
VAR2=VAR4-VAR2      ; Take distance slave travels during accel into account
                    ; so we'll be up to speed at position = 0.2" from end
                    ; of stock.  Then the cut will be made. Initial master
                    ; cycle position will be the negative of the distance
                    ; traveled during slave wait and accel.
FOLMAS31            ; Encoder input #3 is the master for slave axis #1
FMCP(VAR2)          ; Set initial master cycle position to wait length
FOLMD(VAR3)         ; Acceleration to constant ratio will take place over
                    ; VAR3" (1" here) of master travel
FOLRN1
FOLRD1              ; Following ratio is 1 to 1
TRGFNA X1           ; Define a new master cycle on trigger input #1 (TRG-A)
WAIT(IN.25=b1)      ; Suspend program execution until stock is sensed
OUT.6-1             ; Turn on output for saw blade to move down into position
GOWHEN(1PMAS>=0)    ; Wait on first move for start of master cycle.
                    ; This will ensure being at 1:1 ratio at exactly 0.2"
                    ; from end of stock.
$NEWCUT             ; Subroutine DEF for continuous cutting
D,5                 ; Axis #2 will move 5 inches across the stock
MC1                 ; Axis 1 into continuous move mode.
FOLEN1              ; Enable Following on axis #1
VAR5=VAR1           ; Set VAR5 = VAR1 (Snapshot of VAR1)
FMCLEN(VAR5)        ; New master cycle length is cut length
GO1                 ; Start Following move on slave axis #1
```

(Continued)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Continuous Cut-to-Length Program *(Continued)*

```
WAIT(1FS.4=b1)      ; Wait for axis #1 to be in sync with the moving stock
GOx1                ; Once axis #1 is up to speed, move axis #2 across
                    ; stock to make the cut
WAIT(2AS.1=b0)      ; Wait for axis #2 cut to finish
S1                  ; Stop Following move on axis #1
WAIT(MOV=b0)        ; Wait until the move is complete on axis 1
FOLEN0              ; Exit Following mode
OUT.6-0             ; Raise the saw blade
MC0                 ; Axis 1 into preset move mode.
D0,0                ; Move both axes back to home positions
GO11                ; Execute moves on axes 1 and 2 (execution will not
                    ; occur until motion from previous move is complete)
WAIT(MOV=b00)       ; Wait until the moves are complete on axes 1 and 2
OUT.6-1             ; Move saw blade into position for next cut
GOWHEN(1PMAS>=VAR5)  ; Synchronize next move with next master cycle
GOTO NEWCUT         ; Repeat the cut cycle
END                 ; End program definition
```

Technical Considerations for Following

In the introduction to Following (see page 192), the algorithm for 6000 controller Following was briefly discussed. Here we will address some of the more technical aspects of Following:

- Master Position Prediction
- Master Position Filtering
- Following error
- Maximum acceleration and velocity (steppers only)
- Dynamic position maintenance (steppers only)
- Factors affecting Following accuracy
- Preset vs. Continuous Following moves
- Master and slave distance calculations
- Using other features with Following

***Keep in mind that in all cases, the slave position is calculated from a sampled master position.*

Following Status (TFSF, TFS, and FS) bits 17 and 18 indicate the status of master position measurement features:

Bit #	Function (YES =1; NO = Ø)
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.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Master Position Prediction

Master Position Prediction is a technique used to compensate for the fact a slave's position command cannot be calculated and implemented infinitely fast.

The master position prediction mode is enabled by default (FPPEN1) in the Following algorithm, but can be turned off as desired with the FPPENØ command.

Position Sampling Period

Steppers – 2 milliseconds.

Servos – *system update period* (depends on SSFR and INDAX command values – see SSFR command description).

The 6000 controller measures master position once per *position sampling period*, and calculates a corresponding slave position command. This calculation and achieving the subsequent slave commanded position requires 2 sample periods.

If master position prediction mode is disabled (FPPENØ), waiting 2 sample periods results in a slave position lag. That is, by the time the slave reaches the position that corresponds to the sampled master position, 2 sample periods have gone by, and the master may be at a new position. Measured in time, the lag is 2 sample periods. Measured in position, the lag is 2 sample periods ∗ current slave velocity.

For example (stepper controller), suppose the slave is traveling at a speed of 25000 counts per second. If master position prediction mode is disabled (FPPENØ), the slave will lag the master by 100 counts (25000 counts/sec ∗ 4 ms = 100 counts).

By measuring the change in master position over sequential sample periods, the master's present velocity is calculated. The present master velocity and position are used to predict future master position. If master position prediction mode is enabled (FPPEN1, the predicted future master position is used to determine the slave's position command. In this case the slave has no velocity-dependent phase delay. The slave's velocity for a given sample will always be the velocity required to move from its current position to the next calculated position command.

If the master motion is fairly smooth and velocity is not very slow, the measurement of its recent velocity will be very accurate, and a good way of predicting future position. But the master motion may be rough, or the measurements may be inaccurate if there is no filtering (see *Master Position Filtering* below). In this case, the predicted master position and the corresponding slave position command will have some error, which may vary in sign and magnitude from one sample to the next. This random variation in slave position command error results in rough motion. The problem is particularly pronounced if there is vibration on the master.

It may be desirable to disable the master position prediction mode (FPPENØ) when maximum slave smoothness is important and minor phase delays can be accommodated.

If master filtering is enabled (FFILT≠Ø), then the prediction algorithm would be used on the filtered master position, resulting in a smoother slave position command. However, due to the delay introduced by the filtering, the prediction algorithm would not compensate for the total delay in the slave's tracking command. (See also *Master Position Filtering* below.)

Master Position Filtering

Position Sample Period

Steppers – 2 milliseconds.

Servos – *system update period* (depends on SSFR and INDAX command values – see SSFR command description).

The slave axis' position command is calculated at each *position sample period*. This calculation is a function of the master position and the master velocity estimated from the change in master position over 2 position sample periods.

The *Master Position Filter* feature allows you to apply a low-pass filter to the measurement of master position. Master position filtering is 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).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The bandwidth of the low-pass filter is controlled with the FFILT command:

FFILT Setting	Low pass Filter Bandwidth
Ø	∞ (no filtering) – *default setting*
1	120 Hz
2	80 Hz
3	50 Hz
4	20 Hz

NOTE

Increasing the FFILT command value increases the filtering effect (lowers the bandwidth), but at the expense of increasing the phase tracking error (phase lag) of the slave axis. For more information on phase tracking, refer to *Factors Affecting Following Accuracy* below.

When considering whether or how much master position filtering to use, consider the application requirement itself. The application requirements related to filtering can be categorized into these three types:

Type I: If an application requires smooth motion but also high slave tracking accuracy, then a heavy filtering should **not** be used. It should not be used because it may introduce too much phase lag, although the motion may be smooth. In other words, the master axis in the first place should produce very smooth motion and low sensor measurement noise such that a higher level of master filtering is not needed.

Type II: If slave tracking error is not critical but smooth slave axis motion is desired, then you can use a higher level of master filtering to deal with sensor noise or master vibration problems.

Type III: If it is determined that under certain dynamic conditions the master position's oscillatory measurement is purely caused by its vibration motion (noise is insignificant), and it is necessary for the slave to follow such motion, then the filter command should not be used or only use the highest bandwidth (FFILT1).

Following Error

As soon as an axis becomes configured as a slave, the slave's position command is continuously updated and maintained. At each update, the position command is calculated from the current master position and velocity, and the current ratio or velocity of the slave.

Steppers only: This continuously updated position command is used as the target position for the *Dynamic Position Maintenance* feature, described on page 216.

Whenever the commanded position is not equal to the actual slave position, a *Following error* exists. This error, if any, may be positive or negative, depending on both the reason for the error and the direction of slave travel. Following error is defined as the difference between the commanded position and the actual position.

```
Following Error = Commanded position - Actual position
```

If the slave is traveling in the positive direction and the actual position lags the commanded position, the error will be positive. If the slave is traveling in the negative direction and the actual position lags the commanded position, the error will be negative. This error is always monitored, and may be read into a numeric variable (VAR) at any time using the PER command. The error value in slave steps is scaled by SCLD for the axis. This value may be used for subsequent decision making, or simply storing the error corresponding to some other event.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Maximum Velocity and Acceleration (Steppers Only)

The slave's attempt to faithfully follow the master may command velocities and accelerations that the slave axis is physically not able to complete. Therefore, the FMAXV and FMAXA commands are provided to set the maximum velocity and acceleration at which the slave will be allowed to move.

If the slave is commanded to move at rates beyond the defined maximums, the slave will begin falling behind it's commanded position. If this happens, a correction velocity will be applied to correct the position error as soon as the commanded velocity and acceleration fall within the limitation of FMAXV and FMAXA.

The FMAXA and FMAXV commands should be used only to protect against worst case conditions, and should be avoided altogether if they are not needed. If an axis is not able to follow its profile because of limitations imposed by these commands, some correction motion will occur. This is due to the Following error and the resulting *Dynamic Position Maintenance* (see below). If the maximum acceleration (FMAXA value) is set very low, some oscillation about the commanded position may occur because the slave is not allowed to decelerate fast enough to prevent overshoot.

Dynamic Position Maintenance (Steppers Only)

Even while following a master, a slave axis can be in encoder mode with stall detection, position maintenance, and/or deadband wait enabled. In this mode of operation, a difference between actual slave position and the desired slave position may arise just as it may with normal encoder mode moves. Systems with an encoder mounted on a load where mechanical backlash or product stretching is present will be most prone to these desired vs. actual position differences.

An axis becomes a slave by specifying a master with the FOLMAS command. When an axis in encoder step positioning mode (ENC1) becomes a slave axis, it automatically begins the equivalent of position maintenance. This remains true even if the EPMØ (disable position maintenance mode) command is given, and even while the axis is in motion.

There is a very important reason for the continuous dynamic position maintenance. In regular time based moves, the axis has no defined position relationship with anything while it is moving, but it does have a position goal while at rest. Therefore, position maintenance is only meaningful when the axis is not moving. When a slave axis is following a master, there is always a defined position command, calculated from the position of the master. Therefore, position maintenance occurs even while the axis is moving.

The position error in each sample period is usually very small, so only a small correction velocity is added to that required for the slave to follow the master at the commanded ratio. The 6000 product limits the maximum correction velocity to that specified with the EPMV command.

Factors Affecting Following Accuracy

There are additional accuracy requirements of Following applications beyond those of standard positioning. The slave must maintain positioning accuracy while in motion, not just at the end of moves, because it is trying to stay synchronized with the master.

Assuming parameters such as master and slave scaling and ratios have been specified correctly, the overall positioning accuracy for an application depends on several factors:

- Resolution of the master
- Resolution of the slave
- Position sampling accuracy
- Accuracy of the slave motor and drive
- Accuracy of load mechanics
- Master position prediction
- Master velocity relative to master position prediction & master position filtering
- Tuning (servos only)
- Dynamic position maintenance (steppers only)
- Repeatability of the trigger inputs and sensors

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Just as with a mechanical arrangement, the accuracy errors can build up with every link from the beginning to the end. The overall worst case accuracy error will be the sum of all the sources of error listed below. The errors fall into two broad categories, namely, master measurement errors and slave errors. These both ultimately affect slave accuracy, because the commanded slave position is based on the measured master position.

It is important to understand how master measurement errors result in slave position errors. In many applications, master and slave units will be the same (e.g., inches, millimeters, degrees). These applications will require linear speeds or surface speeds to be matched (i.e., a 1:1 ratio). For example, suppose that in a rotary knife application, there were 500 master steps per inch of material, so an error in master measurement of one encoder step would result in 0.002 inches of slave position error.

If the master and slave units are not the same, or the ratio is not 1:1, the master error times the ratio of the application gives the slave error. An example would be a rotary master and a linear slave. For instance, suppose one revolution of a wheel gives 4000 master counts, and results in 10 inches of travel on the slave. The ratio is then 10 inches/revolution. The slave error which results from one step of master measurement error is (1/4000) ∗ 10 inches/revolution = 0.0025 inches.

Resolution of the Master

The best case master measurement precision is the inverse of the number of master steps per user's master unit. For example, if there are 100 master steps/inch, then the master measurement precision is 0.01 inches. Even if all other sources of error are eliminated, slave accuracy will only be that which corresponds to 1 step of the master (e.g., 0.01 inches in the previous example).

Resolution of the Slave

The best case slave precision is the inverse of the number of slave steps per user's position unit. For example, if there are 1000 slave steps/inch, then the slave resolution is 0.001 inches. Even if all other sources of error are eliminated, slave positioning accuracy will only be that which corresponds to 1 step of the slave. This must be at least as great as the precision required by the application.

Position Sampling Accuracy

The position sampling rate for the 6000 controller depends on whether it is a servo or a stepper. The sample period for a stepper is 2 ms. The sample period for a servo is the system update period, which is affected by the current SSFR and INDAX command settings (see SSFR description in the ***6000 Series Software Reference***).

The repeatability of the sampling rate, from one sample to the next, may vary by as much as 20 µs for servos and 100 µs for steppers. This affect may be eliminated by using non-zero master position filter (FFILT) command values. Otherwise, measurement of master position may be off by as much as (20 to 100 microseconds ∗ master speed). This may appear to be a significant value at high master speeds, but it should be noted that this error changes in value (and usually sign) every sample period. It is effectively like a noise of 200-600 Hz; if the mechanical frequency response of the motor and load is much less than this frequency, the load cannot respond to this error.

Accuracy of the Slave Motor and Drive

The precision also depends on how accurately the drive follows its commanded position while moving. Even if master measurement were perfect, if the drive accuracy is poor, the precision will be poor.

In the case of stepper drives, this amounts to the specified motor/drive accuracy.

In the case of servo drives, the better the drive is tuned for smoothness and zero Following error, the better the precision of the positioning. Often, this really only matters for a specific portion of the profile, so the drive should be tuned for zero Following error at that portion.

Accuracy of the Load Mechanics

The accuracy (not repeatability) of the load mechanics must be added to the overall build up of accuracy error. This includes backlash for applications which involve motion in both directions.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Master Position Prediction

The master position prediction mode may be enabled or disabled with the FPPEN command, but each state contributes a different error.

Disabled (FPPENØ): The slave position command is based on a master position that is 2 sample periods old. This means that master measurement error due to disabling the position prediction mode will be (2 sample periods ∗ master speed).

Enabled (FPPEN1): If the position prediction mode is enabled (default setting), its accuracy is also affected by position sampling accuracy, master speed, and master position filtering. The error due to enabling the position prediction mode is about twice that due to sampling accuracy (i.e., 40 to 200 microseconds ∗ master speed). As with the error due to position sampling accuracy, the error due to the position prediction mode being enabled is like a noise on the order of 200-600 Hz, which is not noticed by large loads.

Master Velocity Relative to Master Position Prediction & Master Position Filtering

Variation in Master Velocity: Although increasing master position filtering (increasing the FFILT command value) eliminates the error due to sampling accuracy, it increases the error due to variations in master speed when the master position prediction mode is enabled (FPPEN1).

Most applications maintain a constant master speed, or change very slowly, so this effect is minimal. But if the master is changing rapidly, there may be a significant master speed measurement error. Because predicted master positions are in part based on master speed measurement, the can result in an error in master position prediction mode (FPPEN1). This effect will always be smaller than that due to the master position prediction mode being disabled (FPPENØ).

Phase Tracking : The cost of using the low pass filter (increasing the FFILT command value) is the increase in phase tracking error of the slave's position command. This is an intrinsic characteristic of a low-pass filter. Increasing the filtering (lowering the bandwidth) increases the phase tracking error, or phase lag.

If the master axis is moving at constant velocity, then the delay in the filtered position in terms of time is $\frac{2}{bw}$ sec , where “bw” is the bandwidth in radians/sec. Therefore, at constant velocity, the tracking error of the filtered master position is *velocity* $* \frac{2}{bw}$.

For example, suppose the master is moving at 4000 counts/sec and the master filter bandwidth is 80 Hz, then the filtered master position delay = $\frac{2}{80 * 2\pi}$ = 3.98 msec, and the slave position command tracking error = 4000 ∗ 0.00398 = 15.915 counts.

One important note is that the slave tracking error discussed here refers to the error of the slave's **position command** calculated from the history of the master position signal. The actual slave tracking error also depends on the tuning and the dynamics of the system. For example, one way to eliminate the tracking error introduced by the low pass filter is to use the velocity feedforward gain (SGVF) in the servo loop.

Tuning
(Servos Only)

A servo system's tuning has a direct impact on how well the slave can track the master input. Overshoot, lag, oscillation, etc., can be devastating to Following performance. The best tool to use for tuning the 6000 series controller is Motion Architect's Servo Tuner Module. For tuning instructions, refer to the ***Servo Tuner User Guide*** or to the *Tuning* chapter/appendix your servo controller's installation guide.

Dynamic Position Maintenance
(Steppers Only)

Even when the slave is in motor step mode (ENCØ), there may be one slave step of error inherent to the algorithm. When the slave is in encoder step mode, the user specified position maintenance gain is used to correct position. This gain is expressed as motor steps per second per encoder step error, and has a maximum value of 250 (limited internally). If a value lower than this is used, the position error in encoder steps due to low gain is given by: error = (250/gain) * (encoder resolution/motor resolution).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Repeatability of the Trigger Inputs and Sensors

Some applications may use the trigger inputs for functions like registration moves, GOWHENs, or new cycles. For these applications, the repeatability of the trigger inputs and sensors add to the overall position error.

In the 6000 controller, the position capture from the trigger inputs have approximately 100 µs repeatability, and the sensor repeatability (SR) should be determined, too. Velocity * time = distance, so the error due to repeatability is (SR + 0.0001 seconds) * speed = error. If the sensor repeatability is given in terms of distance, that value can be added directly.

Preset vs. Continuous Following Moves

When a slave performs a preset (MCØ) move in Following mode (FOLEN1), the commanded position is either incremental or absolute in nature, but it does have a commanded endpoint. The direction traveled by the slave will be determined by the commanded endpoint position, and the direction the master is counting.

Let's illustrate this with an example. Assume all necessary set-up commands have been previously issued for our slave (axis #1) and master so that distances specified are in revolutions:

```
FOLRN3      ; Set Following slave-to-master ratio numerator to 3
FOLRD4      ; Set Following slave-to-master ratio denominator to 4
            ; (ratio is 3 revs on the slave to 4 revs on the master)
FOLEN1      ; Enable Following on axis #1
FOLMD10     ; Set preset move to take place over 10 master revolutions
MC0         ; Set slave to preset positioning mode
MA1         ; Set slave to absolute positioning mode
PSET0       ; Set current absolute position reference to zero
D5          ; Set move distance to absolute position 5 revolutions
GO1         ; Initiate move to absolute position 5
```

If the master is stationary when the GO1 command is executed, the slave will remain stationary also. If the master begins to move and master pulses are positive in direction, the slave will begin the preset move in the positive direction. If the master pulses stop arriving before 10 master revolutions have been traveled, the slave will also stop moving, but that GO1 command will not be completed. If the master then starts to count in the negative direction, the slave will follow in the negative direction, but only as far as it's starting position. If the master continues to count negative, the slave will remain stationary. The GO1 command will not actually be completed until the master has traveled at least 10 revolutions in the positive direction from where it was at the time the GO1 command was executed. If the master oscillates back and forth between it's position at the start of the GO1 command to just under 10 revolutions, the slave will oscillate back and forth as well.

The master must be counting in the positive direction for any preset (MCØ) GO1s commanded on the slave to be completed. If mechanics of the system dictate that the count on the source of the master pulses is negative, a minus (-) sign should be entered in the FOLMAS command so that 6000 controller sees the master counts as positive.

Continuous slave moves (MC1) react much differently to master pulse direction. Whereas a preset move will only start the profile if the master counts are counting in the positive direction, a continuous move will begin the ramp to its new ratio following the master in either direction. As long as the master is counting in the positive direction, the direction towards which the slave starts in a continuous move is determined by the argument (sign) of the D command. The slave direction is positive for D+ and negative for D-.

If the master is counting in the negative direction when slave begins a continuous move, the direction towards which the slave moves is opposite to that commanded with the D command. The slave direction is positive for D- and negative for D+.

If the master changes direction during a continuous slave move, the slave will also reverse direction. As with standard continuous moves, the GO1 will continue until terminated by S, K, end-of-travel limits, stall condition, or command to go to zero velocity. As with preset moves, the sign on the FOLMAS command determines the direction of master counting with respect to the direction of actual counting on the master input.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Master and Slave Distance Calculations

The formulas below show the relationship between master move distances and the corresponding slave move distances. These relationships may be used to assist in the design of Following mode moves in which both the position and duration of constant ratio are important.

In such calculations, it is helpful to use SCLMAS and SCLD values which allow the master and slave distances to be expressed in the same units (e.g., inches or millimeters). In this case, many applications will be designed to reach a final ratio of 1:1, and the distances in these figures can be easily calculated.

☞ **HINT**: For a trapezoidal preset slave move with a maximum ratio of 1:1, the master and slave distances during the constant ratio portion will be the same. The slave travel during acceleration will be exactly half of the corresponding master travel, and it will also occur during deceleration.

Master Distance (FOLMD)

If the slave is in continuous positioning mode (MC1), FOLMD is the master distance over which the slave is to accel or decel from the current ratio to the new ratio. If the slave is in preset positioning mode (MCØ), FOLMD is the master distance over which the slave's entire move will take place.

When the Slave is in Continuous Positioning Mode (MC1)

$$D = \frac{FOLMD * (R_2 + R_1)}{2}$$

where:

- FOLMD = Master distance
- R_2 = New ratio (FOLRN ÷ FOLRD)
- R_1 = Current ratio (FOLRN ÷ FOLRD)
- D = Slave distance traveled during ramp

When the Slave is in Preset Positioning Mode (MCØ)

Trapezoidal Slave Moves

$$D_{max} = (FOLMD * R_{max})$$

$$D_1 = \frac{(D_{max} - D)}{2}$$

$$MD_1 = \frac{2 * D_1}{R_{max}}$$

$$D_2 = D - (2 * D_1)$$

$$MD_2 = \frac{D_2}{R_{max}}$$

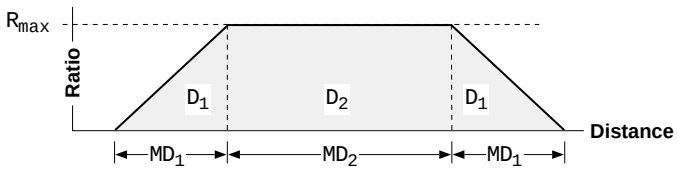
$$FOLMD = (2 * MD_1) + MD_2$$

where:

- FOLMD = Master distance
- MD_1 = Master distance during accel & decel ramps
- MD_2 = Master distance during constant ratio
- D = Total slave preset distance commanded
- D_1 = Slave travel during accel and decel ramps
- D_2 = Slave travel during constant ratio
- D_{max} = Maximum slave distance possible (assuming no accel or decel)
- R_{max} = Maximum ratio = FOLRN ÷ FOLRD

Trapezoidal profile if:

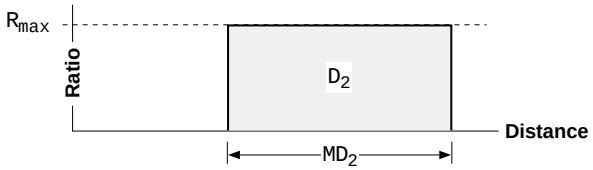
$D = (D_2 + 2 * D_1) < D_{max}$



NOTE ☞ *Profiles depict ratio vs. distance.*

Rectangular profile if:

$D_2 = D$
$D_1 = zero$
$MD_2 = FOLMD$
$MD_1 = zero$



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Triangular Slave Moves

$$D = \frac{R_{peak} * FOLMD}{2}$$

where:

FOLMD = Master distance

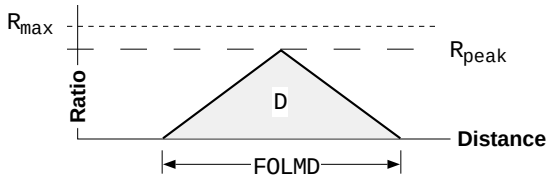
R_{peak} = Peak ratio reached during move

D = Total slave distance

NOTE ☞ *Profile depicts ratio vs. distance.*

Triangular profile if:

$D < 1/2\ D_{MAX}$



Distance Calculation Example

In the example below, the desired travel during constant ratio is already contained in numeric variable #1 (VAR1), and may have been read from thumbwheels or a DATA command. The corresponding slave move distance (D) and FOLMD are calculated as shown:

```
VAR2=2              ; Desired slave travel during accel and decel combined
VAR3=2 * VAR2       ; Required master travel during these ramps
VAR4=VAR1 + VAR2    ; Move distance is constant ratio portion plus ramps
VAR5=VAR1 + VAR3    ; Master travel for entire slave move
FOLMD(VAR5)         ; Establish calculated master travel
D(VAR4)             ; Establish calculated slave travel
GO1                 ; Make desired slave move
```

Similar calculations may be done for a series of continuous move ramps to ratios, separated by GOWHEN for master cycle positions. These ramps may be repeated in a loop to create a continuous cyclical slave profile.

Beware of Roundoff Error (Scaling only)

Some potential for roundoff error exists if the scaling of a move distance or master distance by SCLD and SCLMAS does not result in an integer number of steps. Some additional care must be taken in the segment by segment construction of profiles using ramps to continuous ratio. The 6000 controller maintains a slave position command which is calculated from the commanded constant ratios, the ramps to the new ratios, and the master travel over which these take place. At the end of each ramp or constant ratio portion, this commanded position is calculated to the nearest integer slave step. If the ratios and master travel result in a non-integer slave travel for a segment, the fractional part of that segment's calculated travel will be lost. In a cyclical application, repeated truncations could build up to a significant error. This may be avoided through careful attention to design of the profile.

Using Other Features with Following

The 6000 controller has many features that may be used in the same application as its Following features. In some cases, having configured an axis as a slave with the FOLMAS command will affect, or be affected by, the operation of other features, as described below.

Setups used by FOLMAS (Steppers Only)

The FOLMAS command uses the drive resolution (DRES), the velocity range (PULSE), and the choice of motor step positioning or encoder step positioning (ENC) data to configure an axis as a slave. If encoder step positioning is chosen (ENC1) , the encoder resolution (ERES) data is also used. **In order for this data to be used correctly, these commands must be given before FOLMAS. These commands are not allowed after FOLMAS is given.** After FOLMAS is executed, the MC command may be used to change from incremental to absolute positioning and back, but a change to encoder or motor step positioning (ENC) is not allowed.

S (Stop) & K (Kill) Commands

Stop (S) and Kill (K) commands cause the slave to do a non-Following decel, even if the slave is in Following mode. A stop or a kill, buffered or immediate, will clear a pending GOWHEN condition (clears AS bit #26).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

GO Command

If a slave axis is in Following mode (FOLEN1), moves will ramp to a ratio (set with FOLRN and FOLRD). If it is not in Following mode, moves will ramp to a velocity (V). Switching in and out of Following mode does not change the value for final ratio or final velocity goals, but simply changes which parameter is used as the goal.

Registration Moves

(see page 182)

Registration inputs may be enabled with the INFNCi-H command while an axis is a slave, and registration moves may interrupt either a Following mode move or a time-based move. The registration move itself, however, is always a time based move, and implements the registration velocity with respect to a stationary reference. **Any trigger input may be used as a registration input or for any Following feature which uses triggers, even at the same time.**

Enter/Exit Following Mode While Moving

The FOLENØ command may be executed while a slave is moving at constant ratio. In this case, the current velocity becomes the constant velocity, and the slave may accelerate or decelerate to other velocities. The FOLEN1 command may not be executed while the slave is moving in the non-Following mode. Attempting to do so will result in the error response "MOTION IN PROGRESS".

Pause and Continue

A program may be paused and continued, even if one or more axes is configured as a slave. Those axes do not lose track of the master input, even though motion is stopped. As usual, if a program finishes normally, or if COMEXS is set to zero (COMEXSØ), the program may not be continued. If a program is resumed, partially completed Following moves will be completed; however, the remainder of the move is completed over the entire original master distance (FOLMD value).

TVEL and TVELA Commands

The TVEL and VEL commands have always reported an unsigned value (i.e. magnitude), consistent with the fact that the V command is always positive, even if the move direction is negative. When Following is enabled (FOLEN1), TVEL and VEL report the net magnitude of commanded velocity due to following and/or shifting. For example, if the slave is following in the positive direction at 1 rps, TVEL reports 1.000. If a shift in the negative direction of 1.5 rps is then commanded, TVEL will report 0.5 – still positive, even though the net direction is negative.

By contrast, TVELA and VELA have always been signed. In the above example, TVELA will report -0.5.

TASF, TAS & AS Axis Status Bits

Axis Status (TASF, TAS and AS) bits 1-4 represent the motion status of an axis. Bits 1 and 2 represent the moving and direction status of the net motion of an axis, including the combination of following and shifting.

Bits 3 and 4 represent acceleration and velocity, respectively. With Following disabled (FOLENØ), the accel and velocity can only be due to a commanded time-based profile. With Following enabled (FOLEN1), the net commanded velocity and accel could be due to following the master and/or a simultaneous shift (time-based profile). Bits 3 and 4 only reflect the acceleration and velocity status of the time-based profile, not the combined command.

For example, if the slave is following in the positive direction at 1 rps, TAS reports 1ØØØ. If a continuous shift (FSHFC2) of 0.8 rps in the negative direction is then commanded, TAS will report 1Ø1Ø, while the shaft is decelerating to 0.2 rps. When the continuous shift velocity is reached, TAS will report 1ØØ1.

Changing Feedback Sources **(Servos only)**

Changing feedback sources (SFB) may result in a slave following its own feedback. For this reason, changing feedback sources is not allowed while a master is specified (FOLMAS).

Following and Other Motion

Following motion is initiated with the GO command, just like normal motion. Other motion, which does not depend on the motion of an external master, may not be used while a slave is in Following mode (FOLEN1). These motion types include jog mode, joystick mode, contouring, homing, streaming mode, and linear interpolation (GOL). To use these motion types, Following must be disabled (FOLENØ) — see *Enter/Exit Following Mode While Moving* above for precautions.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Conditional Statements Using PMAS

The master cycle position (PMAS) value may be used in the comparison argument of these commands:

- WAIT & GOWHEN: 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.
- IF, UNTIL, & WHILE: These arguments use the instantaneous PMAS value. 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.

Compiled Motion

Following profiles may be pre-compiled to save processing time. For details, refer to page 166.

Troubleshooting for Following *(see also Chapter 7)*

The table below offers some possible reasons for troubles which may be encountered in achieving the desired slave motion.

Symptom	Possible Causes
Slaves do not follow master	• Improper FOLMAS • Poor connection if master is encoder • Master running backward • No encoder power (when the encoder is selected as the master)
Slave motion is rough	• FFILT command value too low • Unnecessary FPPEN amplifies master roughness
Ratio seems wrong	• FOLRN and FOLRD slave-to-master ratio values are inaccurate, possibly reversed • SCLD or SCLMAS wrong • DRES or ERES wrong for encoder step slaves (steppers only) • Following limited by FMAXV or FMAXA (steppers only)
Slave profile wrong, or un-repeatable	• WAIT used where GOWHEN should be • Too little master travel between GOWHEN and GO1, desired PMAS is missed
Master/slave alignment drifts over many cycles	• Roundoff error due to fractional steps resulting from SCLD or SCLMAS and user's parameters • Ratios and master distances specified result in fractional slave steps covered during ramps, constant ratio
Slave lags Following position **(steppers only)**	• Inhibited by FMAXV • FMAXA *clips* acceleration peaks resulting from attempt to follow rough master
Slave dithers, or oscillates about desired position **(steppers only)**	• EPMG too high for encoder step slave • FMAXA too low for EPMV

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Error Messages

If an illegal programming condition is discovered while programming or executing programs, the 6000 controller responds with an error message. If a program execution error is detected, the program is aborted.

The table below lists all the error messages that relate to Following, and indicates the command and cause that may generate them. These error messages are displayed only if the error level is set to level 4 with the ERRLVL4 command (this is the default setting).

Error Message	Cause
FOLMAS NOT SPECIFIED	No FOLMAS for the axis is currently specified. It will occur if FMCNEW, FSHFC, or FSHFD commands are executed and no FOLMAS command was executed, or FOLMASØ was executed.
INCORRECT DATA	Velocity (V), acceleration (A), or deceleration (AD) command is zero. (used by FSHFC & FSHFD)
INVALID CONDITIONS FOR COMMAND	The FOLMD command value is too small to achieve the preset distance and still remain within the FOLRN/FOLRD ratio. A command phase shift cannot be performed: FSHFD .....Error if already shifting or performing other time based move. FSHFC .....Error if currently executing a FSHFD move, or if currently executing another FSHFC move in the opposite direction. The FOLEN1 command was given while a profile was suspended by a GOWHEN.
INVALID DATA	The parameter supplied with the command is invalid. 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 *Preset Moves* section on page 200)
INVALID FOLMAS SPECIFIED	An illegal master was specified in FOLMAS. A slave may never use its own commanded position or feedback source as its master.
INVALID RATIO	Error if the FOLRN:FOLRD ratio after scaling is > 127 when a GO is executed.
MOTION IN PROGRESS	The FOLEN1 command was given while that slave was moving in a non-Following mode.
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.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Following Commands

Detailed information about these commands is provided in the ***6000 Series Software Reference***.

ERROR .....Enable bit #14 to check for when a GOWHEN condition is already true when a subsequent GO, GOL, FSHFC, or FSHFD command is given.

ERRORP.....If ERROR bit #14 is enabled, a GOSUB branch to the error program occurs if a GOWHEN condition is already true when a subsequent GO, GOL, FSHFC, or FSHFD command is given.

FFILT .....Sets the bandwidth for master position filtering.

FMAXA .....STEPPERS ONLY: Sets the max. acceleration a slave may use while Following.

FMAXV .....STEPPERS ONLY: Sets the max. velocity at which a slave may travel.

FMCLEN.....Defines the length of the master cycle.

FMCNEW.....Restarts new master cycle counting immediately. To make this a trigger-based operation instead of issuing the FMCNEW command, use the TRGFNcx1 command.

FMCP .....Defines the initial position of a new master cycle.

FOLEN .....Enables or disables Following mode.

FOLMAS.....Defines masters for slave axes.

FOLMD .....Defines the master distance over which ratio changes or moves are to take place.

FOLRD .....Establishes the DENOMINATOR ONLY for the max. slave-to-master ratio for a preset move or the final ratio for a continuous move (use in combination with the FOLRN command).

FOLRN .....Establishes the NUMERATOR ONLY for the max. slave-to-master ratio for a preset move or the final ratio for a continuous move (use in combination with the FOLRD command).

FPPEN .....Allows master position prediction to be enabled or disabled.

FSHFC .....Allows continuous advance or retard (shift) of slave position during continuous Following moves.

FSHFD .....Allows preset advance or retard (shift) of slave position during continuous Following moves.

GOWHEN.....A GOWHEN command suspends execution of the next slave move until the specified conditional statement (based on T, IN, PC, PE, PM, PMAS, PSLV, or PSHF) is true. To make this a trigger-based operation instead of issuing the GOWHEN command, use the TRGFNc1x command.

SCLD .....Sets the slave distance scale factor.

SCLMAS.....Sets the master scale factor.

TRGFN .....TRGFNc1xxxxxxx initiates a GOWHEN. Suspends execution of the next slave move until the specified trigger input (c) goes active.
TRGFNcx1xxxxxx allows master cycle counting to be restarted when the specified trigger input (c) goes active.

Status and Assignment Commands:

TASF, TAS & [AS] .....Bit 26 of each axis status is set (1) if a motion profile suspended by a GOWHEN (including TRGFNc1x) is pending on that axis. The bit is cleared when the GOWHEN is true, or when a stop (S) or kill (K) command is executed.

TERF, TER & [ER] .....Bit 14 is set if the GOWHEN condition is already true when a subsequent GO, GOL, FSHFC, or FSHFD command is given. (The corresponding error-checking bits must be enabled with the ERROR command before the error will be detectable.)

TFSF, TFS & [FS] .....Transfers or assigns/compares the Following status or each axis.

TNMCY and [NMCY]...Transfers or assigns the current master cycle number.

TPMAS and [PMAS]...Transfers or assigns the current master cycle position.

TPSHF and [PSHF]...Transfers or assigns the net position shift since constant ratio.

TPSLV and [PSLV]...Transfers or assigns the slave's current commanded position.

TVMAS and [VMAS]...Transfers or assigns the current velocity of the master axis.

WAIT.....A WAIT command suspends program execution until the specified conditional statement (based on PMAS, FS, NMCY, PSHF, PSLV, or VMAS) is true;
WAIT(SS.i=b1) suspends program execution until a trigger input is activated ("i" is the input bit number corresponding to the trigger input—refer to the ***6000 Series Software Reference***).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

CHAPTER SEVEN

7 Troubleshooting

IN THIS CHAPTER

- Troubleshooting basics 228
- Solutions to common problems (problem/cause/remedy table) 228
- Program debug tools
  - Status commands 232
  - Error messages 236
  - Trace mode 239
  - Single-step mode 240
  - Simulating programmable I/O activation 240
  - Simulating analog input activation 242
  - Motion Architect's test panel 242
- Downloading error table (bus-based products only) 243
- Technical support 244
- Product return procedure 244

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Troubleshooting Basics

When your system does not function properly (or as you expect it to operate), the first thing that you must do is identify and isolate the problem. When you have accomplished this, you can effectively begin to resolve the problem.

The first step is to isolate each system component and ensure that each component functions properly when it is run independently. You may have to dismantle your system and put it back together piece by piece to detect the problem. If you have additional units available, you may want to exchange them with existing components in your system to help identify the source of the problem.

Determine if the problem is mechanical, electrical, or software-related. Can you repeat or re-create the problem? Random events may appear to be related, but they are not necessarily contributing factors to your problem. You may be experiencing more than one problem. You must isolate and solve one problem at a time.

Log (document) all testing and problem isolation procedures. Also, if you are having difficulty isolating a problem, be sure to document all occurrences of the problem along with as much specific information as possible. You may need to review and consult these notes later. This will also prevent you from duplicating your testing efforts.

Once you isolate the problem, refer to the problem solutions contained in this chapter. If the problem persists, contact your local technical support resource (see *Technical Support* below).

Electrical Noise

If you suspect that the problems are caused by electrical noise, refer to your 6000 product's ***Installation Guide*** for help.

Solutions to Common Problems

NOTES

- Some hardware-related causes are provided because it is sometimes difficult to identify a problem as either hardware or software related.
- Refer to other sections of this manual for more information on controller programming guidelines, system set up, and general feature implementation. You may also need to refer to the command descriptions in the ***6000 Series Software Reference***. Refer to your product's ***Installation Guide*** for hardware-related issues.

Problem	Cause	Solution
Communication errors (bus-based products).	1. Communication program looking for card at wrong address.	1. Select correct address for communication program.
	2. Address conflict.	2. Refer to the configuration instructions in your Installation Guide (make sure it is matched by the setup in Motion Architect, *if using Motion Architect*)
	3. AT6n00 card not properly seated.	3. Seat board properly in slot. Apply pressure directly over area with gold card edge fingers.
Communication (serial) not operative, or receive garbled characters	1. Improper interface connections or communication protocol.	1. See troubleshooting section in your product's Installation Guide.
	2. COM port disabled.	2.a. Enable serial communication with the E1 command. 2.b. If using RS-485, make sure the internal jumpers are set accordingly (see Installation Guide). Make sure COM 2 port is enabled for sending 6000 language commands (execute the PORT2 and DRPCHKØ commands).
	3. In daisy chain, unit may not be set to proper address.	3. Verify DIP switch settings (see Installation Guide), verify proper application of the ADDR command.
Computer will not boot with AT6nn0 card installed.	1. Interrupt conflict. 2. See problem: *Communication Errors.*	1.a. Turn interrupt DIP switches OFF. 1.b. Refer to the configuration instructions in your Installation Guide (make sure it is matched by the setup in Motion Architect, *if using Motion Architect*)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Problem	Cause	Solution
Direction is reversed. (STEPPERS ONLY)	1. Direction connections to the drive are reversed. (n/a to 610n, 6201)	1. Switch DIR+ with DIR- connection to drive.
	2. Phase of step motor reversed (motor does not move in the commanded direction).	2. Switch A+ with A- connection from drive to motor. SOFTWARE ALTERNATIVE: If the motor (and the encoder if one is used) is reversed, use the CMDDIR1 command to reverse the polarity of both the commanded direction and the polarity of the encoder counts).
	3. Phase of encoder reversed (reported TPE direction is reversed).	3. Swap the A+ and A– connection at the ENCODER connector.
Direction is reversed, servo condition is stable. (SERVOS ONLY)	1. Command output (CMD) connections and feedback device connections or mounting are reversed.	1. Software remedy: Issue the CMDDIR1 command to the affected axis. This reverses the polarity of the commanded direction and the feedback direction so that servo stability is maintained. Hardware remedy: Switch CMD- with the CMD+ connection to drive or valve (if your drive or valve does not accept differential outputs this will not work). You will also have to change the feedback device wiring or mounting so that it counts in same direction as the commanded direction.
Direction is reversed, servo condition is unstable. (SERVOS ONLY)	1. Not tuned properly.	1. Refer to tuning instructions in the *Servo Tuner User Guide*, or in your product's *Installation Guide*.
	2. Phase of encoder reversed or mounting of ANI input or LDT is such that it counts in the opposite direction as the commanded direction.	2. Software remedy: For the affected axis, issue the appropriate feedback polarity reversal command (LDTPOL1 if LDT, ANIPOL1 if ANI analog input, or ENCPOL1 if encoder). Hardware remedy: If using encoder feedback, swap the A+ and A- connections to the 6000 product. If using LDT or ANI feedback, change the mounting so that the counting direction is reversed.
Distance, velocity, and accel are incorrect as programmed.	1. Incorrect resolution setting.	1.a. STEPPERS: Set the resolution on the drive (usually set with DIP switches) to match the 6000 product's DRES command setting (default DRES setting is 25,000 steps/rev). 1.b. Match the 6000 product's ERES command setting (default ERES setting is 4,000 counts/rev) to match the post-quadrature resolution of the encoder. 615n: When using the internal resolver, use ERES4096. ERES values for Compumotor encoders: E Series: ERES4000 OEM Series motors (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 1.c. 6270 only: Check and set the LDT resolution with the LDTRES command.
	2. Pulse width too narrow. (steppers)	2. Set pulse width to drive specifications using the PULSE command.
	3. Wrong scaling values.	3. Check the scaling parameters (SCALE1, SCLA, SCLD, SCLV, PSCLA, PSCLD, PSCLV, SCLMAS) – see also page 83
Erratic operation.	1. Electrical Noise.	1. Reduce electrical noise or move product away from noise source.
	2. Improper shielding.	2. Refer to the Electrical Noise portion of the Technical Reference section in the Compumotor catalog, or in your Installation Guide.
	3. Improper wiring.	3. Check wiring for opens, shorts, & mis-wired connections.
Feedback device (encoder or LDT) counts missing.	1. Improper wiring.	1. Check wiring.
	2. Feedback device slipping.	2. Check and tighten feedback device coupling.
	3. Encoder too hot.	3. Reduce encoder temperature with heatsink, thermal insulator, etc.
	4. Electrical noise.	4a. Shield wiring. 4b. Use encoder with differential outputs.
	5. Encoder frequency too high.	5. Peak encoder frequency must be below 1.6MHz post-quadrature. Peak frequency must account for velocity ripple.
	6. LDT read error (LDT not connected, LDT failure, or LDTUPD setting too fast).	6.a. Connect LDT and issue DRIVE11 command 6.b. Replace faulty LDT and issue DRIVE11 command 6.c. Increase the LDTUPD value and issue DRIVE11 command 6270: To enable an axis (DRIVE11) without an LDT connected, connect GATE+ to GND on the LDT connector.

Following problems — see page 223

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Problem	Cause	Solution
Joystick mode: Motor does not move.	1. Joystick Release input not grounded.	1. Ground Joystick Release input. (n/a to OEM-AT6n00 and single-axis products)
	2. Improper wiring.	2. Check wiring for opens, shorts, and mis-wired connections.
LEDs:	*All other LED states indicate hardware conditions; refer to your product's Installation Guide for details.*	
LED on AT6nn0 PC card is off.	1. No power.	1. Check PC-AT power and check proper card installation in bus slot.
	2. Operating system not downloaded.	2. Download operating system.
"STATUS" LED on AUX board is red. *(n/a to OEM-AT6400)*	1. Operating system not downloaded.	1. Download operating system.
"DISABLED" LED is on	1. Drive was commanded to shut down.	1. Re-enable drive by sending `DRIVE1` command to the affected axis.
	2. Servos: Maximum position error (`SMPER` value) exceeded. Could be caused by disconnected or mismounted feedback device.	2. (verify position error by checking to see if TAS/TASF bit #23 is set) Check feedback device connection and mounting and re-enable drive by sending `DRIVE1` command to the affected axis.
	3. 6270: LDT read error (LDT not connected, LDT failure, or `LDTUPD` setting too fast).	3.a. Connect LDT (or replace faulty LDT) and issue `DRIVE11` command. 3.b. Increase the `LDTUPD` value and issue `DRIVE11` command. 6270: To enable an axis (`DRIVE11`) without an LDT connected, connect GATE+ to GND on the LDT connector.
Motion does not occur.	1. "STATUS" LED is off or red.	1. See LED troubleshooting as noted above.
	2. End-of-travel limits are active.	2.a. Move load off of limits or disable limits by sending the `LHØ` command to the affected axis. 2.b. Software limits: Set `LSPOS` to a value greater than `LSNEG`.
	3. Step pulse too narrow for drive to recognize (steppers only).	3. Set pulse width to drive specifications using the `PULSE` command.
	4. Drive fault level incorrect.	4. Set drive fault level using the `DRFLVL` command (see page 80).
	5. Improper wiring.	5. Check drive fault & limit connections. Steppers: check step and direction connections. Servos: check command and shutdown connections.
	6. Steppers: P-CUT input not grounded. Servos: ENBL input not grounded.	6. Ground the P-CUT or ENBL input connection *(n/a to OEM-AT6n00)*.
	7. Load is jammed.	7. Remove power and clear jam.
	8. No torque from motor.	8. See problem: *Torque, loss of*.
	9. Max. position error (`SMPER` value) exceeded. (servos only)	9. Check to see if TAS/TASF bit #23 is set, and issue the `DRIVE1` command to the axis that exceeded the position error limit.
	10. Drive has activated the drive fault input.	10. Check to see if TAS/TASF bit #14 is set, and check the drive fault level (`DRFLVL`) — see page 80 for appropriate `DRFLVL` settings.
Motor creeps at slow velocity in encoder mode. (steppers only)	1. Encoder direction opposite of motor direction.	1a. Switch encoder connections A+ & A- with B+ & B-. 1b. Switch DIR+ with DIR- connection to drive. (n/a to 610n & 6201)
	2. Encoder connected to wrong axis.	2. Check encoder wiring.
Mouse stops working or serial ports affected (after AT6n00 card is installed).	1. Interrupt conflict. 2. Address conflict.	1. & 2. Refer to the configuration instructions in your product's Installation Guide (make sure it is matched by the setup in Motion Architect, *if using Motion Architect*)
Operating system (AT6nn0) will not download, or download stops part way through.	1. Address conflict. 2. Download error.	1. & 2. Refer to the configuration instructions in your product's Installation Guide (make sure it is matched by the setup in Motion Architect, *if using Motion Architect*)
Power-up Program does not execute.	1. ENBL or P-CUT input is not grounded to GND.	1. Ground the ENBL or P-CUT input to GND and reset.
	2. `STARTP` program is not defined.	2. Check the response to the `STARTP` command. If no program is reported, define the `STARTP` program and reset (see page 15, or refer to the `STARTP` command description).
Program access denied: receive the message `*ACCESS DENIED` when trying to use the `DEF`, `DEL`, `ERASE`, `INFNC`, or `MEMORY` commands. (stand-alone products)	1. Program security function has been enabled (`INFNCi-Q`) and the program access input has not been activated	1.a. Activate the assigned program access input, perform your programming changes, then deactivate the program access input. 1.b. Refer to the instructions on page 15, or to the `INFNC` command description.
Program execution: the first time a program is run, the move distances are incorrect. Upon downloading the program the second time, move distances are correct.	1. Scaling parameters were not issued when the program was downloaded; or scaling parameters have been changed since the program was defined.	1. Issue the scaling parameters (`SCALE1`, `SCLA`, `SCLD`, `SCLV`, `PSCLA`, `PSCLD`, `PSCLV`, `SCLMAS`) before saving any programs.
Program execution: stops at the `INFEN1` command	1. `INFEN1` enables drive fault monitoring, but the drive fault level (`DRFLVL`) command is set incorrectly for the drive being used.	1. Issue the correct `DRFLVL` command for your drive (refer to the `DRFLVL` command). *(n/a to OEM-AT6n00, 610n, 615n, and 6201)*

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Problem	Cause	Solution
Programmable inputs not working.	1. input functions are not enabled — applicable to inputs that are assigned a function other than default (INFNCi-A).	1. Enable programmable input functions with the INFEN1 command.
	2. IN-P (input pull-up) terminal not connected to a power supply.	2. Refer to the connection instructions in your product's Installation Guide.
	3. If external power supply is used, the grounds must be connected together.	3. Connect external power supply's ground to product's ground (GND).
	4. Improper wiring.	4. Check wiring for opens, shorts, and mis-wired connections.
Programmable outputs not working.	1. Output functions are not enabled — applicable to output that are assigned a function other than default (OUTFNCi-A).	1. Enable programmable output functions with the OUTFEN1 command.
	2. Output connected such that it must source current (pull to positive voltage).	2. Outputs are open-collector and can only sink current – change wiring.
	3. OUT-P (output-pull-up) terminal not connected to a voltage source.	3. Refer to the connection instructions in your product's Installation Guide.
	4. If external power supply is used, the grounds must be connected together.	4. Connect external power supply's ground to product's ground (GND).
	5. Improper wiring.	5. Check wiring for opens, shorts, and mis-wired connections.
Runaway (SERVOS ONLY)	1. Direction connections reversed. (if encoder or LDT counts positive when turned clockwise or extended).	1. Switch CMD– with the CMD+ connection to drive or valve. NOTE: The CMD+/– Connection is not differential. Do not connect CMD+ to ground on your drive or valve.
Torque, loss of.	1. Improper wiring.	1. Check wiring to the drive, as well as other system wiring.
	2. No power to drive .	2. Check power to drive.
	3. Drive or valve failed.	3. Check drive or valve status.
	4. Drive faulted.	4. Check drive status.
	5. Shutdown issued to drive or valve.	5. Re-enable drive/valve by sending the DRIVE1 command to the affected axis.
	6. 610n and Zeta drive: Auto standby mode enabled.	6. If more torque is needed at rest, disable auto standby mode: 610n..... issue DAUTOSØ command. Zeta Drive..... Turn off position 1 on DIP switch #2.
Trigger inputs not working.	1. If external power supply is used, the grounds must be connected together.	1. Connect external power supply's ground to product's ground (GND).
	2. Improper wiring.	2. Check wiring for opens, shorts, and mis-wired connections. Refer to your product's Installation Guide for proper wiring.
Velocity & acceleration is incorrect as programmed.	See *Distance* problem noted above.	

Program Debug Tools

After creating your programs, you may need to debug the programs to ensure that they perform as expected. The 6000 controller provides several debugging tools. Detailed descriptions are provided on the following pages.

- **Status Commands**: Use the "Transfer" commands (e.g., TAS, TSS, TIN) to display various controller status information.
- **Error Messages**: You can enable the 6000 controller to display error messages when it detects certain programming errors as you enter them or as the program is run. When the controller detects an error with a command, you can issue the TCMDER command to find out which command caused the error.
- **Trace mode**: Trace a program as it is executing.
- **Single-Step mode**: Step through the program one command at a time.
- **Simulate Programmable I/O Activation**: You can set the desired state of the 6000 controller's 's inputs and outputs via software commands.
- **Simulate Analog Input Activation**: Without an actual voltage present, you can simulate a specific voltage on the 6000 controller's analog input channels using the ANVO command.
- **Motion Architect's Panel Module**: Motion Architect's Panel Module allows you to create custom test panels to verify various system I/O and operating parameters.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Status Commands

Status commands are provided to assist your diagnostic efforts. These commands display status information such as, axis-specific conditions, general system conditions, error conditions, etc.

Checking Specific Setup Parameters

One way to check the conditions that are established with a specific setup command is to simply type in the command name without parameters. For example, type "ERES" to check the encoder resolution setting; the response would look something like: `*ERES4ØØØ`.

Refer to page 78 for a list of most setup parameters and their respective commands.

TIP: To send a status command to the 6000 product during program execution, prefix the command with an exclamation mark (e.g., `!TPER`).

Below is a list of the status commands that are commonly used for diagnostics. Additional status commands are available for checking other elements of your application (see *List of All Status Commands* below). For more information on each status command, refer to the respective command description in the ***6000 Series Software Reference***.

SPECIAL NOTATIONS

* The command has a binary report version (just leave the "F" off when you type it in—e.g., TAS). This is used more by experienced 6000 programmers. Using the binary report command, you can check the status of one particular bit (e.g., The `2TAS.1` command reports "1" if axis #2 is moving or "Ø" if it is not moving.). In the binary report the bits are numbered left to right, 1 through *n*. A "1" in the binary report correlates to a "YES" in the full text report, and a "Ø" correlates to a "NO" in the full text report.

† The command has an assignment/comparison operator that uses the bit status for conditional expressions and variable assignments. For example, the `WAIT(2AS.1=bØ)` command pauses program execution until axis #2's status bit number 1 (`2AS.1`) reports a binary zero value (indicates that the axis is not-moving). See page 6 and page 25 for more information on using assignment and comparison operators in conditional expressions and variable assignments.

TSTAT

Reports general system setup and current conditions.

Sample response for the ZETA6104:

```
*6104 Revision  92-014630-01-4.0 6104
*Participating Axes  1
*Current Motor Position  +0
*Hard Limit Enable  LH3; Soft Limit Enable  LS0
*16 Programs Defined; Scale Enabled 0; Inputs Enabled 1; Outputs Enabled 0
*Drive Resolution  DRES25000
*Encoder Resolution  ERES4000
*Acceleration Scaler  25000
*Distance Scaler  1
*Velocity Scaler  25000
*Acceleration  A10.0000
*Deceleration  AD10.0000
*Velocity  V1.0000
*Distance  D+25000
*Input Configuration  AAAA_AAAA_AAAA_AAAA_AA
*Input State  0100_0000_0000_0000_11
*Output Configuration  AAAA_AAAA_A
*Output State  0000_0000_0
*System Status  1000_1100_0000_0000_0000_1000_0000_0000
*Axis#1 Status  0000_0000_0000_0000_0010_0001_0000_0000
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

TASF

Reports axis-specific conditions. * (TAS) † (AS)

1. Axis is in motion
2. Direction is negative
3. Accelerating
4. At 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) — steppers
11. Position Maintenance on (EPM) — steppers
12. Stall detected (ESTALL) — steppers

13. Drive shutdown occurred
14. Drive fault occurred (enable INFEN1 first)
15. Positive-direction hardware limit hit
16. Negative-direction hardware limit hit

17. Positive-direction software limit (LSPOS) encountered
18. Negative-direction software limit (LSNEG) encountered
19. Within deadband (EPMDB) — steppers only
20. In position (COMEXP) — AT6n00 only

21. In Distance Streaming Mode (STREAM1) — AT6n00 only
22. In Velocity Streaming Mode (STREAM2) — AT6n00 only
23. Position error limit is exceeded (SMPER) — servos
24. Load is within Target Zone (STRGTD & STRGTV)

25. Target Zone timeout occurred (STRGTT)
26. Motion suspended, pending GOWHEN
27. LDT Position Read Error — 6270 only
28. Registration move occurred since last GO

29. RESERVED
30. Pre-emptive (OTF) GO or Registration profile not possible
31. RESERVED
32. RESERVED

TASXF

Reports extended axis-specific conditions. * (TASX) † (ASX)

1. Motor fault occurred — 610n only
2. Low-voltage — 610n only
3. Overtemperature fault — 610n only
4. Drive Fault input is active (hardware state is recognized whether or not the drive is enabled)

TSSF

Reports current system conditions. * (TSS) † (SS)

1. System is ready
2. RESERVED
3. Executing a Program
4. Last command was immediate

5. In ASCII Mode
6. In Echo Mode (ECHO) — stand-alone products
7. Defining a Program (DEF)
8. In Trace Mode (TRACE)

9. In Step Mode (STEP)
10. In Translation Mode — AT6nn0 only
11. Command error (check with TCMDER)
12. Break Point Active (BP)

13. Pause Active (PS or pause input)
14. Wait Active (WAIT)
15. Monitoring On Conditions (ONCOND)
16. Waiting for Data (READ)

17. Loading Thumbwheel Data (TW operator)
18. In External Program Select Mode (INSELP)
19. Dwell in Progress (T command)
20. Waiting for RP240 Data (DREAD or DREADF)

21. RP240 Connected — stand-alone products
22. Non-volatile Memory Error — stand-alone products
23. Gathering servo data — servo products
24. RESERVED

25. Position captured with trigger A (**TRG-A**)
26. Position captured with trigger B (**TRG-B**)
27. Position captured with trigger C (**TRG-C**)
28. Position captured with trigger D (**TRG-D**)

29. Compiled memory partition is 75% full
30. Compiled memory partition is 100% full
31. Compile operation (PCOMP) failed
32. RESERVED

TINOF

Reports extended axis-specific conditions. * (TINO) † (INO)

1. Joystick Auxiliary Input is active
2. Joystick Trigger Input is active
3. Joystick Axes Select Input is active
4. Joystick Velocity Select Input is high (selects JOYVH; low selects JOYVL)

5. Joystick Release Input is active
6. P-CUT input OK (motion not inhibited) — steppers; ENBL input OK (motion not inhibited) — servos
7. NOT USED (always Ø)
8. NOT USED (always Ø)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

TFSF

Reports Following Mode conditions (details on page 193). * (TFS) † (FS)

1. Slave is in a Following ratio move	17. Master Position Prediction Mode enabled (FPPEN)
2. Current ratio is negative	18. Master Position Filtering Mode enabled (FFILT)
3. Slave is changing ratio	19. RESERVED
4. Slave at ratio (constant non-zero ratio)	20. RESERVED
5. FOLMAS Active	21. RESERVED
6. Following Mode enabled (FOLEN)	22. RESERVED
7. Master is moving	23. RESERVED
8. Master direction is negative	24. RESERVED
9. OK to Shift	25. RESERVED
10. Shifting now	26. RESERVED
11. FSHFC-based shift move is in progress	27. RESERVED
12. Shift direction is negative	28. RESERVED
13. Master cycle trigger input is pending	29. RESERVED
14. Mas cycle length (FMCLEN) given	30. RESERVED
15. Master cycle position is negative	31. RESERVED
16. Master cycle number is > 0	32. RESERVED

TERF

Reports error conditions. ** * (TER) † (ER)

1. Stall detected. 1st: Enable Stall Detection (ESTALL).	17. RESERVED
2. Hardware end-of-travel limit encountered. 1st: Enable hard limits (LH).	18. RESERVED
3. Software end-of-travel limit encountered. 1st: Enable hard limits (LH).	19. RESERVED
4. Drive Fault input is active. 1st: Set fault level & enable (DRFLVL & INFEN).	20. RESERVED
5. RESERVED	21. RESERVED
6. A programmable input, defined as a "kill" input (INFNCi-C), is active.	22. RESERVED
7. A programmable input, defined as a "user fault" input (INFNCi-F), is active.	23. RESERVED
8. A programmable input, defined as a "stop" input (INFNCi-D), is active.	24. RESERVED
9. P-CUT input not grounded (steppers); ENBL input not grounded (servos).	25. RESERVED
10. Pre-emptive (OTF) GO or Registration profile not possible.	26. RESERVED
11. Target Zone Settling Timeout Period (STRGTT value) is exceeded.	27. RESERVED
12. Max. position error (SMPER value) is exceeded. — servos only	28. RESERVED
13. RESERVED	29. RESERVED
14. GOWHEN condition already true.	30. RESERVED
15. LDT position read error — 6270 only	31. RESERVED
16. Bad command detected (use TCMDER to identify the bad command)	32. RESERVED

** The error condition will not be reported until you enable the respective error-checking bit with the ERROR command (for details, see page 30 or the ERROR command description). NOTE that when if the error-checking bit is enabled and the error occurs, the controller will branch to the "error" program that you assigned with the ERRORP command.

Other status commands commonly used for diagnostics:

- TDIR ..... Identifies the name and number of all programs residing in the 6000 product's memory. Also reports percent of available memory for programs and compiled path segments.
- TCMDER .... Identifies the bad command that caused the error prompt (?). (see page 238 for details)
- TEX..... Execution status (and line of code) of the current program in progress.
- TIN..... Binary report of all programmable and trigger inputs ("1" = active, "Ø" = inactive). INFNC also reports the state and programmed function of each input. (see page 107 for bit assignments)
- TOUT ..... Binary report of all programmable and auxiliary outputs ("1" = active, "Ø" = inactive). OUTFNC also reports the state and programmed function of each output. (see page 107 for bit assignments)
- TPER ..... Reports the difference between the commanded position and the actual position as measure by the feedback device. Steppers: this command may be used only when the controller is in encoder step mode (ENC1).
- TPM..... Current position of the motor. (steppers only)
- TPE..... Current position of the encoder.
- TFB..... Current position of the feedback device selected with the last SFB command. (servos)
- TPMAS..... Current position of the Following master axis.
- TPSLV..... Current position of the Following slave axis.
- TNMCY..... Current master cycle number.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

List of All Status Commands

SPECIAL NOTATIONS

* The command responds with a binary report. This is used more by experienced 6000 programmers. Using the bit select operator (`.`), you can check the status of one particular bit (e.g., The `2TAS.1` command reports "`1`" if axis #2 is moving or "`Ø`" if it is not moving.). In the binary report, the bits are numbered left to right, 1 through *n*. A "`1`" in the binary report correlates to a "`YES`" in the full text report, and a "`Ø`" correlates to a "`NO`" in the full text report.

Δ The command has a full-text report version (just add an "F" when you type it in—e.g., `TASF`). This makes it easier to check status information without having to look up the purpose of each status bit. (see full-text descriptions on pages 232-234)

† The command has an assignment/comparison operator that uses the bit status for conditional expressions and variable assignments. For example, the `WAIT(2AS.1=bØ)` pauses progress execution until axis #2's status bit number 1 (`2AS.1`) reports a binary zero value (indicates that the axis is not-moving). See page 6 and page 25 for more information on using assignment and comparison operators in conditional expressions and variable assignments.

COMMAND	STATUS SUBJECT
`TANI`	Voltage of ANI inputs *(servo products with ANI option)* †
`TANV`	Voltage of Joystick Analog Inputs (n/a to OEM-AT6n00 and single-axis products) †
`TAS`	Binary Report of Axis Status *Δ†
`TASX`	Binary Report of Axis Status – extended *Δ†
`TCA`	Value of Captured ANI Input †
`TCMDER`	Command Error (view command that caused the error prompt)
`TCNT`	Hardware Counter Value †
`TDAC`	Digital-to-Analog (DAC) Voltage (Servos) †
`TDIR`	Program Directory and Available Memory
`TDPTR`	Data Pointer Status †
`TER`	Error Status *Δ†
`TEX`	Program Execution Status †
`TFB`	Position of Selected Feedback Devices †
`TFS`	Binary Report of Following Status *Δ†
`TGAIN`	Current Value of Active Servo Gains
`TIN`	Binary Report of Status of Programmable Inputs *†
`TINO`	Status of Joystick Inputs and P-CUT or ENBL *Δ†
`TINT`	Binary Report of Interrupt Status *†
`TLABEL`	Defined Labels (names of)
`TLDT`	Position of LDT †
`TLIM`	Binary Report of Hardware Status of All Limit Inputs †
`TMEM`	Memory Usage (partition and available memory)
`TNMCY`	Master Cycle Number †
`TOUT`	Binary Report of Status of Programmable Outputs *†
`TPANI`	Position of ANI Inputs *(servos with ANI option)* †
`TPC`	Commanded Position *(servos)* †
`TPCA`	Captured ANI Input Position *(servos with ANI option)* †
`TPCC`	Captured Commanded Position *(servos)* †
`TPCE`	Captured Encoder Position †
`TPCL`	Captured LDT Position *(6270)* †
`TPCM`	Captured Motor Position *(steppers)* †
`TPE`	Position of Encoder †
`TPER`	Position Error †
`TPM`	Position of Motor *(steppers)* †
`TPMAS`	Position of Master Axis †
`TPROG`	Contents of a Program
`TPSHF`	Net Position Shift †
`TPSLV`	Current Commanded Position of the Slave Axis †
`TREV`	Firmware Revision Level
`TSGSET`	Servo Gain Sets
`TSEG`	Number of Free Segment Buffers †
`TSS`	Binary Report of System Status *Δ†
`TSTAT`	Statistics
`TSTLT`	Settling Time
`TTIM`	Time Value †
`TUS`	User Status *†
`TVEL`	Current Commanded Velocity †
`TVELA`	Current Actual Velocity †
`TVMAS`	Current Velocity of the Master Axis †

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

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 (includes contouring) 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 FOLMAS command was executed, or FOLMASØ 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. See memory allocation guidelines on page 12.
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): • FOLMD command value is too small to achieve the preset distance and still remain within the FOLRN/FOLRD ratio. • A command phase shift cannot be performed: FSHFD ..... Error if already shifting or performing other time based move. FSHFC ..... Error if currently executing a FSHFD move, or if currently executing another FSHFC move in the opposite direction. • The FOLEN1 command was given while a profile was suspended by a GOWHEN.
INVALID CONDITIONS FOR S_CURVE ACCELERATION–FIELD *n*	Average (A_{avg}) acceleration or deceleration command (e.g., AA, ADA, HOMAA, HOMADA, etc.) with a range that violates the equation $1/2A_{max} \le A_{avg} \le A_{max}$. ($A_{max}$ is the maximum accel or decel command—e.g., A, AD, HOMA, HOMAD, etc.)
INVALID DATA	Data for a command is out of range Following (these conditions can cause an error during Following): • The parameter supplied with these commands is invalid: 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 *Preset Moves* on page 200)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Error Responses *(continued)*

Error Response	Possible Cause
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/second.
MOTION IN PROGRESS	Attempting to execute a command not allowed during motion (see *Restricted Commands During Motion* on page 18). 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 (includes contouring) path 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, SGVFN) and SMPER are not allowed if SFBØ is selected.
NOT ALLOWED IN PATH	Compiled Profile (includes contouring) 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.
PATH ALREADY MOVING	Compiled Profile (includes contouring) path compilation error
PATH NOT COMPILED	Attempting to execute a individual axis profile or multi-axis contouring path that has not been compiled
PATH RADIUS TOO SMALL	Contouring path compilation error
PATH RADIUS ZERO	Contouring path compilation error
PATH VELOCITY ZERO	Contouring path compilation error
STRING ALREADY DEFINED	A string (program name or label) with the specified name already exists
STRING IS A COMMAND	Defining a program or label that is a command or a variant of a command
UNDEFINED LABEL	Command issued to product is not a command or program name
WARNING: POINTER HAS WRAPPED AROUND TO DATA POINT 1	During the process of writing data (DATTCH) or recalling data (DAT), the pointer reached the last data element in the program and automatically wrapped around to the first datum in the program
WARNING: ENABLE INPUT INACTIVE	Servo controllers only: **ENBL** input is no longer connected to ground (**GND**)
WARNING: PULSE CUT INPUT ACTIVE	Stepper controllers only: **PCUT** input is no longer connected to ground (**GND**)
WARNING: DEFINED WITH ANOTHER TW/PLC	Duplicate I/O in multiple thumbwheel definitions

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Identifying Bad Commands

To facilitate program debugging, the Transfer Command Error (TCMDER) command allows you to display 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, and error status bit #16 (reported in the TERF, TER and ER 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 than16, 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
>
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Trace Mode

You can use the Trace mode to debug a program. The Trace mode allows you to track, command-by-command, the entire program as it runs. The 6000 controller will display all of the commands as they are executed. For stand-alone controller users, program tracing is also available on the RP240 display (see page 134).

The example below demonstrates the Trace mode.

Step 1 Create a program called `prog1`:

```
DEF prog1           ; Begin definition of program prog1
A10                 ; Acceleration is 10
AD10                ; Deceleration is 10
V5                  ; Velocity is 5
L3                  ; Loop 3 times
GOSUB prog3         ; Gosub to program #3 (prog3)
LN                  ; End the loop
END                 ; End definition of program prog1
```

Step 2 Create program `prog3:`

```
DEF prog3           ; Begin definition of program prog3
D50000              ; Sets the distance to 50,000
GO1                 ; Initiates motion
END                 ; End definition of program prog3
```

Step 3 Enable the Trace Mode:

```
TRACE1              ; Enables the Trace mode
```

Step 4 Execute the program `prog1`: (each command in the program is displayed as it is executed)

```
EOT13,10,0          ; Set End-of-Transmission characters to <cr>,<lf>
RUN prog1           ; Run program prog1
```

The response will be:

```
*PROGRAM=PROG1        COMMAND=A10.0000
*PROGRAM=PROG1        COMMAND=AD10.0000
*PROGRAM=PROG1        COMMAND=V5.0000
*PROGRAM=PROG1        COMMAND=L3
*PROGRAM=PROG1        COMMAND=GOSUB PROG3  LOOP COUNT=1
*PROGRAM=PROG3        COMMAND=D50000  LOOP COUNT=1
*PROGRAM=PROG3        COMMAND=GO1  LOOP COUNT=1
*PROGRAM=PROG3        COMMAND=END  LOOP COUNT=1
*PROGRAM=PROG1        COMMAND=LN  LOOP COUNT=1
*PROGRAM=PROG1        COMMAND=GOSUB PROG3  LOOP COUNT=2
*PROGRAM=PROG3        COMMAND=D50000  LOOP COUNT=2
*PROGRAM=PROG3        COMMAND=GO1  LOOP COUNT=2
*PROGRAM=PROG3        COMMAND=END  LOOP COUNT=2
*PROGRAM=PROG1        COMMAND=LN  LOOP COUNT=2
*PROGRAM=PROG1        COMMAND=GOSUB PROG3  LOOP COUNT=3
*PROGRAM=PROG3        COMMAND=D50000  LOOP COUNT=3
*PROGRAM=PROG3        COMMAND=GO1  LOOP COUNT=3
*PROGRAM=PROG3        COMMAND=END  LOOP COUNT=3
*PROGRAM=PROG1        COMMAND=LN  LOOP COUNT=3
*PROGRAM=PROG1        COMMAND=END
```

The format for the Trace mode display is:

```
Program Name  ...  Command  ...  Loop Count     or
Program Name  ...  Command  ...  Repeat Count   or
Program Name  ...  Command  ...  While Count
```

Step 5 Exit the Trace Mode.

```
TRACE0              ; Disables the Trace mode
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Single-Step Mode

The Single-Step mode allows you to execute one command at a time. Use the STEP command to enable Single-Step mode. To execute a command, you must use the !# sign. By entering a !# followed by a delimiter, you will execute the next command in the sequence. If you follow the !# sign with a number (*n*) and a delimiter, you will execute the next *n* commands. The Single-Step mode is demonstrated below (using the programs from the Trace mode above).

Step 1 Enable the Single-Step Mode:

```
STEP1        ; Enables Single Step Mode
```

Step 2 Enable the Trace Mode and begin execution of program prog1:

```
TRACE1       ; Enables the Trace mode
RUN prog1    ; Run program called prog1
```

Step 3 Execute one command at a time by using the !# command:

```
!#           ; Executes one command
```

The response will be:

```
*PROGRAM=PROG1          COMMAND=A10.0000
```

Step 4 To execute more than one command at a time, follow the !# sign with the number of commands you want executed:

```
!#3          ; Executes three commands
```

The response will be:

```
*PROGRAM=PROG1          COMMAND=AD10.0000
*PROGRAM=PROG1          COMMAND=V5.0000
*PROGRAM=PROG1          COMMAND=L3
```

To complete the sequence, use the # sign until all the commands are completed (!#16 would complete the example).

Step 5 To exit Single-Step mode, type:

```
STEP0        ; Disables Single Step Mode
```

Simulating I/O Activation

If your application has inputs and outputs that integrate the 6000 controller with other components in your system, you can simulate the activation of these inputs and outputs so that you can run your programs without activating the rest of your system. Thus, you can debug your program independent of the rest of your system.

There are two commands that allow you to simulate the input and output states desired. The INEN command controls the inputs and the OUTEN command controls the outputs.

Servo Products

The INEN command has no effect on the trigger inputs (**TRG-A** through **TRG-D**) when they are configured as *trigger interrupt* (position latch) inputs with the INFNCi-H command.

The OUTEN command has no effect on the auxiliary outputs (**OUT-A** through **OUT-D**) when they are configured as *output-on-position* outputs with the OUTFNCi-H command.

You will generally use the INEN command to cause a specific input pattern to occur so that a program can be run or an input condition can become true. Use the OUTEN command to simulate the output patterns that are needed, and to prevent an external portion of your system from being initiated by an output transition. When you execute your program, the OUTEN command overrides the outputs and holds them in a defined state.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Input and Output Bit Patterns Vary by Product

Input and output bit patterns vary by product. For example, the 6250's input pattern comprises 24 general-purpose inputs (bits 1-24) and 2 trigger inputs (bits 25 & 26); in contrast, the OEM6250's input pattern comprises 16 general-purpose inputs (bits 1-16) and 2 trigger inputs (bits 17 & 18). To ascertain the bit pattern for your product, consult the I/O bit pattern table on page 107.

Outputs

The following steps describe the use and function of the OUTEN command.

Step 1 Display the state of the outputs with the TOUT command:

```
TOUT          ; Displays the state of the outputs
```

The response will be:

```
*TOUT0000_0000_0000_0000_0000_0000_00
```

Display the function of the outputs with the OUTFNC command:

```
OUTFNC        ; Displays the state of the outputs
```

The response will be:

```
*OUTFNC1-A PROGRAMMABLE OUTPUT - STATUS OFF
*OUTFNC2-A PROGRAMMABLE OUTPUT - STATUS OFF
*OUTFNC3-A PROGRAMMABLE OUTPUT - STATUS OFF
.
.
*OUTFNC26-A PROGRAMMABLE OUTPUT - STATUS OFF
```

Step 2 Disable outputs 1 - 4, leave them in the ON state.

```
OUTEN1111     ; Disable outputs 1-4, leave them in ON state
OUTFNC        ; Displays the state of the outputs
```

The response will be:

```
*OUTFNC1-A PROGRAMMABLE OUTPUT - STATUS DISABLED ON
*OUTFNC2-A PROGRAMMABLE OUTPUT - STATUS DISABLED ON
*OUTFNC3-A PROGRAMMABLE OUTPUT - STATUS DISABLED ON
.
.
*OUTFNC26-A PROGRAMMABLE OUTPUT - STATUS OFF
```

Step 3 Change the output state using the OUT command. The status of all outputs, including auxiliary outputs, is displayed. The output bit pattern varies by product. To determine the bit pattern for your product, refer to the OUTEN command description.

```
OUT1010       ; Activates outputs 1 and 3, deactivates outputs 2 and 4
```

Display the state of the outputs with the OUTFNC command.

```
OUTFNC        ; Displays the state of the outputs
```

The response will be:

```
*OUTFNC1-A PROGRAMMABLE OUTPUT - STATUS DISABLED ON
*OUTFNC2-A PROGRAMMABLE OUTPUT - STATUS DISABLED ON
*OUTFNC3-A PROGRAMMABLE OUTPUT - STATUS DISABLED ON
.
.
*OUTFNC28-A PROGRAMMABLE OUTPUT - STATUS OFF
```

Notice that output 2 and output 4 have not changed state because the output (OUT) command has no effect on disabled outputs.

Step 4 To re-enable the outputs, use the OUTEN command.

```
OUTENEEEE     ; Re-enables outputs 1-4
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Inputs

The steps below describe the use and function of the INEN command. You can use it to cause an input state to occur. The inputs will not actually be in this state but the 6000 controller treats them as if they are in the given state and will use this state to execute its program.

Step 1 This program will wait for an input state to occur and will then make a preset move:

```
INFNC1-A       ; Input #1 is has no function
INFNC2-A       ; Input #2 is has no function
INLVL00        ; Set input #1 and #2 active level to low
DEF prog8      ; Begin definition of program prog8
A100           ; Acceleration is set to 100
AD100          ; Deceleration is 100
V5             ; Velocity is 5
D25000         ; Distance is 25,000
WAIT(IN=b11)   ; Waits for the input state to be 11
GO1            ; Initiate motion
END            ; End definition of program prog8
```

Step 2 Enable the Trace mode so that you can view the program as it is executed:

```
TRACE1         ; Enables the trace mode
```

Step 3 Execute the program:

```
RUN prog8      ; Runs program prog8
```

Step 4 The program will execute until the WAIT(IN=b11) command is encountered. The program will then pause, waiting for the input condition to be satisfied. Simulate the input state using the INEN command. Inputs with an E value are not affected. Note that the input bit pattern varies by product. To determine the bit pattern for your product, refer to the INEN command description.

```
!INEN11        ; Disables inputs 1 and 2, leaving them in the ON state
```

The motor will now move for 25000 steps.

Step 5 Deactivate the input simulation:

```
INENEE         ; Re-enables inputs 1 and 2
```

Simulating Analog Input Channel Voltages

Without actually applying any voltage, you can test any command or function that references the voltage on the analog channels found on the **JOYSTICK** connector. For example, ANVO1.2,1.6,1.8 overrides the hardware analog input channels 1 through 3 as follows: 1.2V on channel 1, 1.6V on channel 2, and 1.8V on channel 3.

The ANVO values will be recognized only for those analog input channels for which ANVOEN is set to 1 (e.g., Given ANVOENØ11, the ANVO values 1.6V and 1.8V will be referenced for analog channels 2 and 3 only.).

Another application for the ANVO command may be to use it in an ERRORP program to override the analog input voltage in response to a fault.

Motion Architect's Panel Module



User instructions are provided in the *Motion Architect User Guide*.

Motion Architect's Panel module provides tools to create custom test panels to test your programs and monitor the following:

- I/O (programmable I/O, analog I/O, limits)
- Motion (motor and feedback device position, velocity)
- Status (axis, system, interrupt, user-defined, Following)
- Timer and counter values
- Terminal (direct communication with the product, to check for error messages, etc.)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Downloading Error Table *(bus-based controllers only)*

Error	Description	Reason/Corrective Action
1	Operating System File Not Found	The operating system specified, or the default operating system (if unspecified) could not be found by the AT6nnn.EXE loader program. Put the AT6nnn.OPS in the same directory as the AT6nnn.EXE file.
2	Invalid Operating System File	The operating system specified, or the default operating system (if unspecified) is not a valid operating system or is corrupted. Re-install the operating system from the original disk.
3	Unexpected EOF	An EOF character was received during the download. Re-install the operating system from the original disk.
4	Invalid Port Address	The port address specified while downloading is invalid. Use another address setting (768 ≤ port ≤ 1024 in increments of 8).
5	Unknown Option	An unknown option was specified on the AT6nnn.EXE command line.
6	Base Port Address Greater than 1024	The base port address is too high. Specify an address between 768 and 1024 decimal with the /PORT= parameter.
7	Base Port Address Less than 255	The base port address is too low. Specify an address between 768 and 1024 decimal with the /PORT= parameter.
8	Base Port Address Not a Multiple of 8	The base port address is not a multiple of 8. Specify a valid address with the /PORT= parameter.
9	Modified Download Requested	A partial download was requested on the command line.
10	Card Controller Error	The card controller did not respond as expected. Verify that you are downloading to the correct address. Make sure there are no other peripheral cards (network adapters, bus mouse, etc.) at the same address. Try changing the card address.
11	Card Not found	The card did not respond as expected. Verify that you are downloading to the correct address. Make sure there are no other peripheral cards (network adapters, bus mouse, etc.) at the same address. Try changing the card address.
12	Reading Card Rev	The card appeared to be working as expected until the revision was requested. Verify that you are downloading to the correct address. Make sure there are no other peripheral cards (network adapters, bus mouse, etc.) at the same address. Try changing the card address.
13	Waiting for Data Ready	The card did not respond when expected. Verify that you are downloading to the correct address. Make sure there are no other peripheral cards (network adapters, bus mouse, etc.) at the same address. Try changing the card address.
14	Purging Data Out Buffer	The card output buffer could not be emptied. Verify that you are downloading to the correct address. Make sure there are no other peripheral cards (network adapters, bus mouse, etc.) at the same address. Try changing the card address.
15	Waiting for Data Input Buffer Empty	The card did not respond to the data sent to it. Verify that you are downloading to the correct address. Make sure there are no other peripheral cards (network adapters, bus mouse, etc.) at the same address. Try changing the card address.
16	Time-out Waiting for Processor Startup	The card did not respond as expected. The green LED on the back of the PC-card should be on for this error to occur. Verify that you are downloading to the correct address. Make sure there are no other peripheral cards (network adapters, bus mouse, etc.) at the same address. Try changing the card address. Use a fresh copy of the operating system from the disk that was shipped with the card. If the green LED on the back of the card flashes briefly during download of the operating system, the card may need repair.
17	CRC Error	The CRC value calculated during download is not the same as stored with the operating system. Either the file is corrupted on disk, or was corrupted during download. Try a fresh copy of the operating system. If your computer has a Turbo switch, switch it to low speed.
18	Operating System Rev not Compatible with Loader Rev	The operating system being downloaded is not compatible with the AT6nnn.EXE file (downloader) being used. Use the same downloader on the diskette with the operating system.
19	Incompatible Card ROM rev	The card ROMS and the AT6nnn.EXE file (downloader) are incompatible. If you are using a new downloader, obtain a new set of ROMS from the factory.
20	Card Read Error (bad compare)	The downloader is unable to communicate reliably with the card. Try switching to 8-bit mode on the card, switching out of Turbo mode on your PC, or using a different address.
21	Card Read Error (outbuf)	The downloader is unable to empty the output buffer. There may be an address conflict with another board. Try a different address.
23	Card ROMS - Unsupported Option Requested	The card ROMS do not support the option specified on the command line. Obtain a ROM update from the factory.
24	NULL Error	

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Technical Support

For solutions to your questions about implementing 6000 product software features, first look in this manual. Other aspects of the product (command descriptions, hardware specs, I/O connections, graphical user interfaces, etc.) are discussed in the respective manuals listed above in *Reference Documentation* (see page ii).

If you cannot find the answer in this documentation, contact your local Automation Technology Center (ATC) or distributor for assistance.

If you need to talk to our in-house application engineers, please contact us at the numbers listed on the inside cover of this manual. (The phone numbers are also provided when you issue the HELP command to the 6000 controller.)

Product Return Procedure

If you must return your 6000 Series product to affect repairs or upgrades, use this procedure:

Step 1 Get the serial number and the model number of the defective unit, and a purchase order number to cover repair costs in the event the unit is determined by the manufacturers to be out of warranty.

Step 2 Before you return the unit, have someone from your organization with a technical understanding of the 6000 Series product and its application include answers to the following questions:

- What is the extent of the failure/reason for return?
- How long did it operate?
- Did any other items fail at the same time?
- What was happening when the unit failed (e.g., installing the unit, cycling power, starting other equipment, etc.)?
- How was the product configured (in detail)?
- What, if any, cables were modified and how?
- With what equipment is the unit interfaced?
- What was the application?
- What was the system environment (temperature, enclosure, spacing, unit orientation, contaminants, etc.)?
- What upgrades, if any, are required (hardware, software, user guide)?

Step 3 Call for a return authorization. Refer to the Technical Support phone numbers provided on the inside cover of this manual. The support personnel will also provide shipping guidelines.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

I N D E X

16-bit mode 39
6000 DOS Support Disk 37
8-bit mode 39

A

absolute position
  absolute positioning mode 88
  absolute zero position 88
  establishing 88
  status 88
acceleration
  change on the fly 87, 178
  maximum (steppers) 216
  S-curve profiling 146
  scaling 84, 196
  units of measure 87
access to RP240 functions 137
accuracy
  Following, factors affecting 216
  position capture 113, 182
active damping 82
address
  conflict 228
  daisy chain (RS-232) 72
  downloading to 39
  multi-drop (RS-485) 75
ANA output, use of 142
analog inputs (joystick)
  override voltage 140, 242
  status from RP240 136
analog output (servos)
  add a square wave (dither) 101
  aux. output (ANA) 142
  offset 102
  setting max & min limits 102
ANI input 142
  4-20 mA feedback 142
  position 136
anti-resonance 82
application examples
  cam profiling (using compiled Following) 172, 175
  continuous cut-to-length 211
  continuous phase shift 202
  contouring 161
  electronic gearbox 204
  grinding (using distance streaming) 149
  packaging (using on-the-fly motion) 180
  PLC 127
  preset phase shift 203
  registration 184, 185, 186
  scaling setup 86
  servo setup 103
  spindle (using compiled motion) 170
  stamping (using compiled Following) 174
  teaching data points 122
  trackball 205
  using ANI inputs 127
  using joystick 127
  using programmable I/O 127
  using RP240 127
application program
  developing your own 42
  downloading from 40
  downloading from DOS prompt 41
  terminal emulation 41
arc segments 153, 157
assignment & comparison operators 6
  used in conditional expressions 24
assignment of master and slave 194
assumptions, skills required to implement features ii
AT6n00 reference iii
auxiliary programmable outputs 119
axes, participating 79
axis moving status 233
axis scaling 83
axis status 135, 233
  extended 233
axis status, relative to Following 222

B

background polling loop 68
base address 43
BCD program select input 110
before you start programming iii
begin program definition (DEF) 9
binary variable (VARB) 25
binary variables (VARB) 18, 22
bit patterns, programmable I/O 107
bit select operator (.) 5, 7
bitwise operations (and, or, not, etc.) 22
Boolean operations 21
Borland Turbo C 2.0 38
Borland Turbo PASCAL 5.0 38
branching 23
  conditional 24
  unconditional 23
buffers
  buffered commands
    control execution of 16
    executed during motion 89, 178
    stored in a program 8
  command 89, 111
  input 63
  output 42, 63, 67
  ring 68
bus communication registers 43
bus-based controllers
  device driver (DLL) 51
  interrupt path 63
  operating system, *see operating system*

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

C

C (tangent) axis, contouring 154, 159
C program, downloading from 40
CAD-to-motion software iv, 2
capture positions (motor, encoder, ANI, LDT, commanded) 112
carriage return, command delimiter 5
case sensitivity 5
CCW end-of-travel limits, *see limits, end-of-travel*
center joystick position 138
center specified arcs 158
change summary i
characters
  command delimiters 5
  comment delimiter 5
  field separators 5
  limit per line 5
  neutral (spaces) 5
checksum 33
circles 158
circular buffers 68
closed-loop operation, steppers 95
COM ports, controlling 70
commanded direction polarity
  servos 101
  steppers 97
commanded position
  absolute position reference 88
  slave, tracking error 215
  status 234
commands
  4.x (new), see Change Summary
  buffer 89, 111
    after pause 111
    after stop 111
  command buffer execution
    after end-of-travel limit (COMEXL) 17
    after in-position signal (COMEXP) 17
    after kill (COMEXK) 16
    after pause/continue input (COMEXR) 17
    after stop (COMEXS) 17
    continuous (COMEXC) 16
  command value substitutions 6
  delimiters 3, 5
  errors in programming 238
  executed during motion 178
  Following (list of) 225
  immediate 3, 89
  restricted execution during motion 18
  setup command list 78
  status 232
  syntax 3
comment delimiter 3, 5
communication
  AT bus registers 43
  controlling multiple serial ports 70
  DDE server 50
  DLLs 51
  DOS support software iv, 37
    downloading 39
    terminal emulation 41
  interrupt 63
  Motion Architect iii
communication *(continued)*
  OCX toolkit 62
  problems 228
  RS-232C daisy-chaining 72
  RS-485 multi-drop 75
  via RP240 terminal 131
compiled motion 163
  compare with on-the-fly motion changes 169
  Following profiles 166
  related commands 169
  sample applications 170
CompuCAM™ iv, 2
computer interrupts 63
conditional branching 24, 28
conditional expression examples 25
conditional GO 186
  pending trigger input 113, 189
conditional looping 24, 28
conditional statements using PMAS 209
configuration
  6104 internal drive 82
  address 73
  closed-loop stepper setup 95
  DAC output limits 102
  disable drive on kill 82
  dither 101
  drive fault level 80
  drive resolution 81
  end-of-travel limits 90
  feedback device polarity 100
  Following setup 194
  homing 91
  inputs 108
  jogging 114
  joystick 138
  memory allocation 12
  outputs 116
  participating axes 79
  positioning modes 87
  scaling 83
  servo control signal offset 102
  servo setup 98
  setup commands, list 78
  setup program 14
  start/stop velocity 82
  step pulse (steppers) 81
  target zone 105
  thumbwheel 129
  tuning 99
  variable arrays 120
continue (!C) 111
continue execution on pause/resume (COMEXR) 17, 111
continue execution on stop (COMEXS) 17, 111
continue input 111
continue key on RP240 130
continue, effect on Following motion 222
continuous command execution mode (COMEXC)
  effect in continuous positioning mode 89
continuous cut-to-length application 211
continuous positioning mode 87, 89
  Following 199, 219
    distance calculations 220
contouring 153
  acceleration 155
    scaling 84
  affected by drive resolution 81, 154
  affected by pulse width 81
  arcs 157
  C axis 153, 154, 159
  circles 158
  compiling a path 160
  deceleration 155
    scaling 84
  defining a path 153
  distance, scaling 85
  effected by pulse width 154
  endpoints 155
  executing a path 161
  helical interpolation 155, 159
  lines 156
  local coordinates 156
  memory allocation 12
  outputs along path 160
  P (proportional) axis 153, 154, 159
  participating axes 154
  programming errors 161
  scaling 83
  segment boundary 158
  stall 159
  velocity 155
    scaling 84
  work coordinates 156
controlling multiple serial ports 70
correction, velocity 216
counter 96
  value, assignment/comparison (CNT) 26
counting, master cycle, *see master, master cycle*
current standby mode 82
current waveform 82
CW end-of-travel limits, *see limits, end-of-travel*

D

DAC output, limiting 102
daisy-chain 72
  including RP240 74
data
  fields, in command syntax 4
  read from serial port or PC bus 26
  read from the RP240 26
  teach to variable arrays 120
datapoint, streaming 148
DDE6000 (DDE server) 50, 144
deadband
  joystick 138
  position maintenance 95
  stall 96
debounce time
  position capture 112
  program select input 110
  programmable inputs 109
  registration input 182

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

debugging tools 231
analog channel voltages, simulating 242
error messages 236
from RP240 134
I.D. bad command 238
I/O activation, simulating 240
problem/cause/solution table 228
single-step mode 240
status commands 232
trace mode 239
deceleration
change on the fly 87, 178
S-curve profiling 146
scaling 84
scaling, Following 196
units of measure 87
defining a program 9
delimiters
command 5
comment 5
detecting a stall 96
stall indicator output 118
device address, *see address*
device driver, dynamic link library (DLL) 51
direction, changes in compiled motion 168
distance
calculations, Following 220
compiled moves 167
change on the fly 87, 178
fractional step truncation 85, 197
registration 182
scaling, *see scaling*
streaming 148
status 233
units of measure 87
dithering motors & hydr valves & 101
DLL 51, 144
function descriptions for Visual Basic 52
function descriptions for Visual C++ 58
DOS support software iv, 37
download program (from Motion Architect) 10
downloading application programs from DOS prompt 41
downloading, operating system
from application programs 40
from DOS 39
C 40
error 39, 243
methods 39
PASCAL 40
from Motion Architect iii
LED status 230
problems 230
drive 126
6104 internal drive config. 82
fault input
active level (DRFLVL) 80
enable 80
status 233
on/off status 137
resolution 81
effect on contouring 81, 154
effect on Following 221
drive *(continued)*
shutdown
LED status 230
on kill 82, 111
dwells & direction changes, compiled motion 168
dynamic data exchange (DDE) server 50
dynamic link libraries (DLLs) 51
dynamic position maintenance 216
effect on accuracy 218

E

electrical noise, *see Installation Guide*
electronic gearbox application 204
electronic I/O devices 130
electronic viscosity 82
electronics concepts ii
enable input
as safety feature 126
status 233
enable or disable Following 198
status 193
while moving 222
encoder
encoder step mode (ENC1) 95
feedback for steppers 95
polarity reversal
servos 100
steppers 97
position 136
capture 112
resolution 95
setup example
servos 103
steppers 96
Z channel 91
end point
contouring 155
linear interpolation 152
end program/subroutine/path definition (END) 9
end-of-move settling 105
end-of-travel limits, *see limits, end of travel*
error
clearing 31
error handling 30
downloading errors 243
error level for daisy chains 72
related to safety 126
error messages 236
Following specific 224
Following (PER) 215
position (dynamic pos. maintenance) 216
program, assignment 30
status 234
example programs ii
executing programs *see program, execution options*

F

fast status registers 43
customizing 48
fault output 118
feedback about this manual i
feedback source, changing (servos) 222
feedrate override 141
field separator (,) 5
filtering, master position, *see master, master position filtering*
firmware revision, check with RP240 137
Following 192
commands, list of 225
compiled Following profiles 166
conditions used in conditional expressions 27
enable or disable 198
status 193
while moving 222
error 215
performance considerations 193
prerequisites to Following motion 194
Ratio Following introduction 192
set-up parameters 194
status 193, 234
technical considerations 213
fractional step truncation 197
full or half step start/stop velocity 82

G

gains, servo tuning 99
view & edit from RP240 133
general purpose interrupt 68
global command identifier (@) 5
GOSUB 23
GOTO 23
GOWHEN 186
cleared by stop or kill 221
error condition 234
using PMAS 209, 223
via trigger input 113, 189
GUI development tools 144

H

hard limit, *see limits, end-of-travel*
hardware interrupts 64
head pointer 68
helical interpolation 155, 159
Help (technical support services) iv, 244
hexadecimal identifier (h) 22
Homing 91
home limit, *see limits, home*
status 233
zeroing the absolute position 91
host computer operation 143

I

I/O activation (simulation) 240
I/O device interface 128
IF 24
using PMAS 209, 223
IM32 module 129
immediate commands 3, 89
not stored in programs 8
immediate data read from RP240 27
immediate stop 89

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

in position input, effect on command execution 17
in position output 117
incremental positioning mode 88
initial master cycle position 208
input buffer is 256 bytes 50
input ring buffer 68
input-buffer-is-empty interrupt 68
inputs 107
analog 138, 141
ANI option 142
application example 127
overriding 140
bit patterns 107
drive fault 126
active level (DRFLVL) 80
status 233
enable (ENBL) 126
status 233
encoder, *see encoder*
end-of-travel limits 89, 90, 126
home limits 91
jogging 114
joystick 138
kill 111
limits 90
no function 109
one-to-one program select 115
pause/continue 111
PLC 130
program security 116
program select 110, 115
programmable 106, 128
bit patterns 107
debounce time 109
enable functions (INFEN), effect on system performance 33
function assignments 108
interrupt to PC-AT 114
kill 89
operand (IN) 25
pause/continue, effect on command buffer 17
polarity 106
problems 231
program security 15
simulating activation 240
status 106, 108, 136
stop 89
update rate 106
pulse cut (P-CUT) 126
status 233
stop 111
thumbwheel 129
triggers
debounce time 109
position capture 112
problems 231
programmed functions (TRGFN) 113
update rate 106
user fault 112, 126
interface options 127
interface routines 42
interpolation
circular/contouring *(see contouring)*
linear *(see linear interpolation)*
interrupts
conflict 228
enable within AT6nnn 63
enable within PC-AT 64
general-purpose 67
how to use (step-by-step) 65
important considerations 67
input-data-buffer-empty 67
interrupt driver 69
interrupt path 63
interrupt service routine (ISR) 63
maskable 64
output-buffer-has-data 67
PC-AT 63
programmable input function 114
program (ON conditions) 29
status-update 67
terminal emulator 68
vectors 64

J

jerk, acceleration, reducing 146
jogging 114
negative direction (INFNCi-aK) 114
positive direction (INFNCi-aJ) 114
RP240 jog mode 134
speed select hi/low (INFNCi-aL) 114
status 134
velocity 134
high (JOGVH) 114
low (JOGVL) 114
joystick
analog inputs 138
status from RP240 136
voltage override 140
application example 122, 127
center deadband 138
center voltage 138
Daedal JS6000 138
inputs 139
status 233
status from RP240 136
interface 127, 138
problems 230
select input 138
velocity resolution 138
voltage override 140, 242
JS6000 joystick 138
JUMP 23

K

kill
assigned input function 89, 111
effect on command execution 16
effect on drive 82, 111
effect on Following motion 221
kill on stall 96

L

labels ($) 23
LabVIEW VIs 144
last motion segment, compiled motion165
LDT
distance scaling 85
position 136
read error 229, 234
LEDs 230
RP240 130
left-to-right math 5
length, master cycle 207
limit to DAC output 102
limits
end-of-travel 90, 126
effect on command buffer and program execution 17
status 233
used as basis to activate output 118
home 91
status 136
line feed, command delimiter 5
line segments 153, 156
linear interpolation 152
acceleration scaling 84
distance scaling 85
end point 152
velocity scaling 84
local coordinate system 156
lockout distance for registration 182
logical operators 25
loops
conditional 27
unconditional 23

M

master
definition of 194
status 193
direction, status of 193
distance
move calculations 220
programming (FOLMD) 198
master cycle
concept 207
counting 207
restart 208
status 207
length 207
number 209
position 207
assignment/comparison (PMAS) 209
initial 208
rollover 207, 223
synchronizing 210
transfer (TPMAS) 209
status 193
master input (A.K.A.) 192
master position filtering 193, 214
effect on accuracy 218
effect on phase tracking 218
effect on position accuracy 217
status 193, 213
master position prediction
mode193, 214
effect on accuracy 218
status 193, 213
move profiles 199
moving, status of 193
ratio to slave 198
status 193
resolution 217
scaling 196
velocity 193, 214

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

master/slave daisy-chain 72, 74
mathematical operations 19
maximum acceleration (steppers) 216
maximum position error
  establishing 99
  output to indicate when exceeded 118
maximum velocity (steppers) 216
memory
  allocation 12
    compiled motion 163
    contouring 153
  cleared on bad checksum 33
  expanded (-M option) 13
  locking 15, 116
  non-volatile 33
  status 13
menus, RP240 133
messages, error 236
Microsoft ASSEMBLY 5.1 38
Microsoft C 6.0 38
Microsoft QuickBASIC 4.5 38
motion
  Following motion status 194
  motion control concepts ii
  parameters used in conditional expressions 25
  pre-compiled profiles, *see compiled motion*
  restrictions while in Following 222
  rough 223
  synchronize
    conditional GOs 186
    registration 182
    triggered conditional GOs 186
    triggered start of master cycle 186
    with PMAS 210
Motion Architect iii, 2
  manual is on web (www.compumotor.com) ii
  programming scenario 8
  servo tuner option 99
  setup program tool 14
Motion Builder iv
Motion OCX Toolkit 62, 144
Motion Toolbox 144
motor
  dithering 101
  fault, status (610n) 233
  inductance setting 82
  motor step mode (ENCØ) 95
  position 26
    capture 112
  static torque setting 82
  stiction 101
move completion criteria 105
moving/not moving status 117, 233
multi-drop, RS-485 75
multiple serial ports, controlling 70

N

negative-direction end-of-travel limits, *see limits, end-of-travel*
neutral characters 5
no function input 109
noise, electrical, *see installation guide*
numeric variables 18
  in conditional expression 25
NWHILE 28

O

OCX toolkit 62, 144
on conditions (program interrupts) 29
  effect on system performance 33
on-line help for Windows 2
on-line manuals ii
on-the-fly motion changes 87, 178
  compare with compiled motion 169
one-to-one program select input 115
operating system, downloading
  from DOS 39
    errors 243
  from Motion Architect iii
  LED status 230
  problems 230
operator interface
  create your own GUI 144
  host computer program 143
  RP240 remote panel 130
operators
  assignment & comparison 6
    used in cond. expressions 24
  bit select 7
  bitwise 8
  logic 8
  math 8
  relational 8
output buffer
  output-buffer-has-data interrupt 68
  prevent from filling up (reading) 42
output on position 119
output ring buffer 68
outputs 107
  activate on position 116, 119
  ANA 142
  bit patterns 107
  configuration 116
  contouring path 160
  fault output 118
  limit encountered 118
  maximum position error exceeded 118
  moving/not moving (in position) 117
  OUT-A 119
  OUT-B 119
  OUT-C 119
  OUT-D 119
  program in progress 118
  programmable 106, 128
    bit pattern 107
    enable functions (OUTFEN), effect on system performance 33
    function assignments 116
    operand (OUT) 25
    polarity 106
    problems 231
    simulating activation 240
    status 106, 117, 136
    update rate 106
  shutdown 126
  stall indicator 118
  update rate 106

P

P axis, contouring 153, 154, 159
participating axes 79
  for contouring 154
partitioning memory 12
PASCAL program, downloading from 40
password, RP240 137
paths *(see contouring)*
pause active, status 233
pause key on RP240 130
pause, effect on Following motion 222
pause/continue input 111
  effect on motion & program execution 17
PC-AT interrupts 63
  programmable input function 114
performance, effects on 33
performance, Following 193
phase, shift 201
  status 201
phase, tracking 218
PLC interface 130
  application example 127
point-to-point move 88
polarity
  ANI inputs 100
  commanded direction
    causes reversed direction 229
    servos 101
    steppers 97
  encoder
    servos 100
    steppers 97
  feedback device, servos 100
  LDT 100
  programmable I/O 106
position
  absolute 88
    establish with PSET 88
  accuracy, Following 217
  ANI 136, 142
    capture 112
  capture 112
    encoder 233
    motor 233
    registration 182
  commanded 136
    capture 112
  commanded position calculation, Following 215
  dynamic position maintenance 216
    effect on accuracy 218
  encoder 26, 136
    capture 112
  error 136, 216, 229
    exceeded max. limit status 233
    max. allowable 99, 126
  incremental 88
  LDT 26, 136
    read error 234
  master, *see master*
  motor 26
    capture 112
  position maintenance, steppers 95
  positioning modes 87
    change on the fly 87, 178
  sampling period 193
    effect on Following accuracy 217
  slave 201
  status via RP240 136
  used to activate output 119
  zeroed after homing 91

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

positive-direction end-of-travel limits, *see limits, end-of-travel*
potentiometer, joystick 138
power-up start program (STARTP) 15
  clear RP240 menus 132
  Following setup commands 195
  will not execute 230
pre-emptive GOs, *see on-the-fly motion*
prediction of master position, *see master, master position prediction*
preset positioning mode 88
  Following 200, 219
    distance calculations 220
priority levels, PC-AT interrupts 64
product naming convention iii
profiling, custom 145
program
  branch
    conditionally 24
    unconditionally 23
  buffer 9
  comments 3, 5
  debug tools 231
  definition 9
  download from Motion Architect 10
  editing in Motion Architect 2, 11
  error handling 30
  error responses 236
  examples, using ii
  execution
    controlling 16
    from Motion Architect 11
    from RP240 menu 134
    options 14
    status 233
  flow control 23
  interrupts 29
  labels ($) 23
  loop
    conditionally 24
    unconditionally 23
  memory allocation 12
  power-up program 15
  program development scenario 8
  programming guidelines 1
  saving to disk 10
  security 15
  setup (configuration) program 14
  storage 12
program in progress 118
programmable inputs, *see inputs, programmable*
programmable outputs, *see outputs, programmable*
programming
  ASSEMBLY language iv
  BASIC language iv
  C language iv
  contouring errors 161
  contouring examples 161
  debug tools 231
  debugging via RP240 134
  defining a program 9
  develop your own DOS application program 42
  downloading programs
    errors 243
    from DOS 39
    from Motion Architect iii, 10
programming *(continued)*
  edit in Motion Architect 11
  error messages 236
  error programs 32, 126
  examples, using ii
  executing programs
    from Motion Architect 11
    options 14
  languages
    DOS Support Disk sub-directories 38
    samples 38
  PASCAL language iv
  preparing to program iii
  program development scenario 8
  program security 116
  program selection
    BCD 110
      debounce time 110
    one-to-one 115
  sample programs on DOS support disk 38
  sample programs provided ii
  saving programs to disk 10
  set-up program 14, 79
  skills required ii
  storing programs 12
proportional axis, contouring 153, 154, 159
pulse width 81
  effect on contouring 154
  effect on streaming 149
pulse-cut input
  as safety feature 126
  status 233

R

radius specified arcs 157
radius tolerance specifications 157
ratio of slave to master 198
  status 193
reading
  from the AT6nnn 50
  inputs and outputs 106
    thumbwheel data 129
  output buffer 42
reentrance 64
reference documentation ii
registers
  communication 43
    fast status 43
  hardware 96
registration 182
  effect on Following 222
  sample application 184, 185, 186
  status 183
related documentation ii
relational operators 25
REPEAT 24
repeatability, Following
  position sampling rate 217
  sensors 219
  trigger inputs 219
resetting the controller 79
  via the RP240 137
resolution
  drive 81
    effect on Following 221
  encoder 95
  joystick 138
  master 217
  slave 217
responses, error 236
restart master cycle counting 113, 189, 208
restricted commands during motion 18
return procedure 244
revision levels (product, DSP & RP240) 137
ring buffers 68
rollover of master cycle position 207, 209, 223
rough Following motion 223
RP240 130
  access security, password 137
  application example 127
  COM port setup 71, 131
  connection verified 233
  data read 26
  front panel description 130
  in daisy chain 74
  menu structure 133
  send text via the connector 131
RS-232C communication
  daisy-chaining 72
  RP240 connector 131
RS-485 multi-drop 75
runaway motor 231

S

safety features 126
sample programs ii
saving programs to disk 10
scaling 83, 87
  acceleration & deceleration 84
  distance 85
  effect on system performance 33
  master 196
  slave 196
  velocity 84
security, program 15, 116
segment boundary 158
segment, definition of 13
serial ports, controlling 70
servo
  data gathering, status 233
  sampling rate, effect on system performance 33
  setup 98
    code examples 103
  tuning 99
  updates rates, *see change summary*
Servo Tuner™ iv, 2, 99
set-up commands 78
set-up program 14, 79
  bus-based controllers 15
  stand-alone controllers 15
settling time, actual 105
shift
  continuous 201
  preset 201
  status 193, 201

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

shift left to right (>>) 22
shift right to left (<<) 22
shutdown 126
  LED status 230
  on kill 82, 111
side-by-side editor and terminal 2
simulating analog input voltages 242
single-shot registration 182, 185
single-step mode 134, 240
  status 233
slave
  commanded position 201
    Following error 215
  conditional go 186
  definition of 194
  distance
    move calculations 220
    scaling 196
  motor/drive accuracy 217
  move profiles 199
  ratio to master 198
    status 193
  resolution 217
  scaling 196
  shift, *see shift*
soft limit, *see limits, end-of-travel*
space (neutral character) 5
square-wave signal (dither) 101
stall deadband 96
stall detection 96
  stall indicator output 118
stand-alone operation 127
start-up program (STARTP) 15
  examples for servos 103
  include setup parameters 79
  problems 230
start/stop velocity 82
statistics, controller config, status 232
status
  absolute position 88
  assigned to binary variable 19
  axis 135, 136
    extended 233
    relative to Following 222
  command error 238
  commands
    diagnostics related 232
    list of 235
  compiled motion 164
  error conditions 234
  fast status registers 43
  Following 193, 234
  GOWHEN 187
  inputs 108, 136
    enable (ENBL) 233
    pulse cut (P-CUT) 233
  joystick inputs 136, 233
  LEDs 230
  limits 136, 233
  master cycle number (TNMCY) 209
  master cycle position (TPMAS) 209
  motion 25, 233
  OTF profiling conditions 179
  outputs 117, 136
  pause 233
  position 234, 235
    captured 112, 233
    RP240 display 136
  program execution 233
status *(continued)*
  registration 183
  RP240 displays 135, 136
  setup parameters, basic 78
  statistics (TSTAT) 232
  system 135, 136, 233
  wait 233
step output pulse width 81
stiction 101
stop
  assigned input function 89
  effect on Following motion 221
  effect on program execution 17, 111
  stop key on RP240 130
storing programs in controller memory 12
storing variable data to arrays 120
streaming mode 148
  affected by pulse width 81
  streaming data (SD) 148
    datapoint 148
string variables 18
subroutine, definition of 8
substitutions, command values 6
support software
  CompuCAM iv
  DDE6000™ 50, 144
  DLLs 51, 144
  DOS support disk 38
  Motion Architect iii
  Motion OCX Toolkit™ 62, 144
  Motion Toolbox™ 144
  Servo Tuner iv
support, technical iv
synchronizing motion
  conditional GOs (GOWHEN) 186
  Following (slave-to-master) 210
  registration 182
  trigger functions
    conditional GO 189
    start new master cycle 189
syntax 4
  guidelines 5
system performance 33
system status 135, 233
system update period (servos) 193

T

tail pointer 68
tangent axis, contouring 153, 154, 159
target zone 105
  affects moving/not moving output 117
  status (within zone) 233
  timeout error 105
    status 233
teach mode 120
technical considerations for Following 213
technical support iv, 244
terminal emulation 2
  DOS support disk programs 41
  interrupt-driven 68
  Motion Architect iii
testing
  AT6n00 test program (TEST.EXE) 38
  program debug tools 231
  test programs, Motion Arch. 2, 242
thumbwheels
  application example 127
  connections 129
  TM8 module 129
  use of 129
timed data streaming, *see streaming*
timeout, target zone 105
  status 233
trace mode 134, 239
  status 233
trackball application 205
trigger inputs
  I/O bit pattern 107
  position capture 112
    status 233
  programmed
    conditional GO (GOWHEN) 113, 189
    restart master cycle counting 113, 189, 208
    status 193
  repeatability 219
trigonometric operations 20
troubleshooting
  common problems and solutions 228
  controller statistics 232
  debug tools 231
  downloading, error messages 243
  ENBL status 233
  error messages 236
  Following 223
  I.D. bad command 238
  methods 228
  P-CUT status 233
  status commands 232
  test panels, Motion Architect 242
truncation, distance 85, 196
tuning 99
  effect on Following accuracy (servos) 218

U

UNTIL 24
  using PMAS 209, 223
update rate, programmable I/O 106
user fault input 112, 118, 126
user interface options 127
  GUI development tools 144
user programs, memory allocation 12

V

value substitution, command fields 6
valve
  dithering 101
  stiction 101
variables
  binary 18, 22
    in conditional expression 25
  conversion between binary & numeric 19
  numeric 18
    in conditional expression 25
    teach data 120
  string 18
  variable arrays 120
  view & edit from RP240 133

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

velocity
change on the fly 87, 178
correction 216
maximum (steppers) 216
range due to PULSE (steppers) 221
resolution 138
scaling 84, 196
start/stop 82
streaming 148
status 233
TVEL & TVELA responses relative to Following 222
units of measure 87
Visual Basic support files 52
Visual C++ support files 58

W

WAIT 24
compared to GOWHEN 188
status 233
using PMAS 209, 223
watchdog timer 126
web site (www.compumotor.com) ii
WHILE 24, 28
using PMAS 209, 223
Windows-based application development 51
work coordinate system 156
writing to the AT6nnn 50

X-Z

X-Y linear interpolation 152
XON/XOFF, controlling 70
Z channel 91, 94
zero position after homing 91

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

6000 Series Programmer's Guide

General Programming

See also, page 1

Command syntax..... page 3
Programming scenario (using Motion Architect)..... page 8
Program flow control (loops, branches, WAITs).....page 23
Error handling (programmed responses)..... page 30
Error messages.....page 236
Debugging your programs.....page 231
Programmable I/O bit patterns..... page 107
Scaling.....page 83
Variables..... page 18

Common problems (see also problem/cause/solution table on page 228):

- No motion. Possible causes include:
  - P-CUT input or ENBL input must be grounded to allow motion. Check status with TINOF command.
  - Drive is disabled. Enable with DRIVE command. Make sure the drive fault level (DRFLVL) is correct (see pg. 80)
  - End-of-travel limit active (check with TASF, bits 15-18)
- Programmable input (INFNC) functions do not work.
  - Enable with the INFEN1 command.

Command list and descriptions (see *6000 Series Command Reference*)

Technical Assistance: Phone numbers are reported with the HELP command, and are listed on the inside cover of this document.

Status Commands

See also, page 232

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 *(see also page 235)*

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 (see page 30). If the error-checking bit is enabled and the error occurs, the controller will branch to the ERRORP program.
TINOF.....Status of P-CUT or ENBL input and joystick inputs.
TIN.....Active status of each programmable & trigger input. Bit assignments vary by product—refer to page 107.
TOUT.....Active status of each programmable & auxilliary output. Bit assignments vary by product—refer to page 107.
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.

Basic Setup

See also, page 78

Setup Parameter *	Command	See pg.
Participating Axes	INDAX	79
Memory (status with TDIR & TMEM)	MEMORY	12 & 80
Drive Setup		80
Drive Fault Level	DRFLVL	
Drive Resolution	DRES	
Step Pulse Width (steppers)	PULSE	
Start/Stop Velocity (steppers)	SSV	
Drive Disable on Kill (servos)	KDRIVE	
ZETA6104 Drive Setup:		
Motor inductance	DMTIND	
Motor static torque	DMTSTT	
Activate damping	DACTDP	
Anti-resonance	DAREN	
Electronic viscosity	DELVIS	
Automatic current reduction	DAUTOS	
Axis Scaling		83
Enable scaling factors	SCALE	
Acceleration scaling factor	SCLA	
Distance scaling factor	SCLD	
Velocity scaling factor	SCLV	
Positioning Mode		87
Continuous or preset	MC	
Preset: absolute or incremental	MA	
End-of-travel limits		90
Hardware – enabled	LH	
Hardware – deceleration	LHAD	
Hardware – s-curve decel (servos)	LHADA	
Hardware – active level of input	LHLVL	
Software – enabled	LS	
Software – deceleration	LSAD	
Software – s-curve decel (servos)	LSADA	
Software – negative direction limit	LSNEG	
Software – positive direction limit	LSPOS	
Homing		91
Acceleration	HOMA	
S-curve acceleration (servos)	HOMAA	
Deceleration	HOMAD	
S-curve deceleration (servos)	HOMADA	
Backup to home	HOMBAC	
Final approach direction	HOMDF	
Stopping edge of switch	HOMEDG	
Home switch active level	HOMLVL	
Velocity	HOMV	
Velocity of final approach	HOMVF	
Home to Z channel input	HOMZ	
Closed-loop Stepper Setup (stepper products only)		95
Motor (drive) resolution	DRES	
Encoder resolution	ERES	
Encoder/motor step mode select	ENC	
Position maintenance	EPM	
Stall detection	ESTALL	
Kill on stall detected	ESK	
Stall deadband	ESDB	
Use encoder as counter	CNTE	
Encoder polarity	ENCPOL	
Commanded direction polarity	CMDDIR	
Servo Setup (servo products only)		98
Tuning parameters	(see page 99)	
Feedback source selection	SFB	
Update rates	SSFR	
Maximum position error	SMPER	
DAC output limit, maximum	DACLIM	
DAC output limit, minimum	DACMIN	
Dither, amplitude	SDTAMP	
Dither, frequency ratio	SDTFR	
Feedback polarity, encoder	ENCPOL	
Feedback polarity, ANI input	ANIPOL	
Feedback polarity, LDT	LDTPOL	
Commanded direction polarity	CMDDIR	
Servo control signal offset	SOFFS	
Servo control signal offset, negative	SOFFSN	
Setpoint window, distance	SSWD	
Setpoint window, gain set	SSWG	
Target Zone (end-of-move settling criteria)		105
Target zone mode enable	STRGTE	
Target distance zone	STRGTD	
Target velocity zone	STRGTV	
Target settling timeout period	STRGTT	
Programmable Input Functions (TIN for binary status report)		106 & 108
Enable input functions	INFEN	
Define input functions	INFNC	
Input active level	INLVL	
Input debounce	INDEB	
Trigger input functions	TRGFN	
Programmable Output Functions (TOUT for binary status report)		106 & 116
Enable output functions	OUTFEN	
Define output functions	OUTFNC	
Output active level	OUTLVL	
Variable Arrays (teaching variable data)		120
Initialize numeric variable for data	VAR	
Define data program and program size	DATSIZ	
Set data pointer & establish increment	DATPTR	
Reset data pointer to specific location	DATRST	

***** You can check the status of each parameter by entering the setup command without any command fields (e.g., INFNC). Some parameters are also reported with the TSTAT and TASF status commands.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP