ACR Motion Controllers

88-025604-01G

ACR Command Language Reference

Effective: September 2008

```
SET 34   : REM Turn on OUTPUT 34 (Camera Process)
DWL 5    : REM Wait 5 seconds (for Camera Process to finish)
CLR 34   : REM Turn off OUTPUT 34 (Camera Process)
X12 Y6   : REM Move to Position 2
SET 34   : REM Turn on OUTPUT 34 (Camera Process)
DWL 5    : REM Wait 5 seconds (for Camera Process to finish)
CLR 34   : REM Turn off OUTPUT 34 (Camera Process)
X18 Y6   : REM Move to Position 3
SET 34   : REM Turn on OUTPUT 34 (Camera Process)
DWL 5    : REM Wait 5 seconds (for Camera Process to finish)
CLR 34   : REM Turn off OUTPUT 34 (Camera Process)
X18 Y12  : REM Move to Position 4
SET 34   : REM Turn on OUTPUT 34 (Camera Process)
DWL 5    : REM Wait 5 seconds (for Camera Process to finish)
CLR 34   : REM Turn off OUTPUT 34 (Camera Process)
X12 Y12  : REM Move to Position 5
```

Parker

ENGINEERING YOUR SUCCESS.

User Information

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

Technical Assistance

Contact your local automation technology center (ATC) or distributor.

North America and Asia
Parker Hannifin
5500 Business Park Drive
Rohnert Park, CA 94928
Telephone: (800) 358-9070 or (707) 584-7558
Fax: (707) 584-3793
Email: emn_support@parker.com
Internet: http://www.parkermotion.com

Germany, Austria, Switzerland
Parker Hannifin
Postfach: 77607-1720
Robert-Bosch-Str. 22
D-77656 Offenburg
Telephone: +49 (0) 781 509-0
Fax: +49 (0) 781 509-176
Email: sales.hauser@parker.com
Internet: http://www.parker-emd.com

Europe (non-German speaking)
Parker Hannifin plc
Electromechanical Automation, Europe
Arena Business Centre
Holy Rood Close
Poole
Dorset, UK
BH17 7BA
Telephone: +44 (0) 1202 606300
Fax: +44 (0) 1202 606301
Email: support.digiplan@parker.com
Internet: http://www.parker-emd.com

Italy
Parker Hannifin
20092 Cinisello Balsamo
Milan, Italy via Gounod, 1
Telephone: +39 02 6601 2478
Fax: +39 02 6601 2808
Email: sales.sbc@parker.com
Internet: http://www.parker-emd.com

Parker Automation

Technical Support E-mail

emn_support@parker.com

Table of Contents

Important User Information

It is important that motion control equipment is installed and operated in such a way that all applicable safety requirements are met. It is your responsibility as an installer to ensure that you identify the relevant safety standards and comply with them; failure to do so may result in damage to equipment and personal injury. In particular, you should study the contents of this user guide carefully before installing or operating the equipment.

The installation, setup, test, and maintenance procedures given in this guide should only be carried out by competent personnel trained in the installation of electronic equipment. Such personnel should be aware of the potential electrical and mechanical hazards associated with mains-powered motion control equipment—please see the safety warnings below. The individual or group having overall responsibility for this equipment must ensure that operators are adequately trained.

Warning — High-performance motion control equipment is capable of producing rapid movement and very high forces. Unexpected motion may occur especially during the development of controller programs. KEEP WELL CLEAR of any machinery driven by stepper or servo motors. Never touch any part of the equipment while it is in operation.

This product is sold as a motion control component to be installed in a complete system using good engineering practice. Care must be taken to ensure that the product is installed and used in a safe manner according to local safety laws and regulations. In particular, the product must be positioned such that no part is accessible while power may be applied.

This and other information from Parker Hannifin Corporation, its subsidiaries, and authorized distributors provides product or system options for further investigation by users having technical expertise. Before you select or use any product or system, it is important that you analyze all aspects of your application and review the information concerning the product in the current product catalog. The user, through its own analysis and testing, is solely responsible for making the final selection of the system and components and assuring that all performance, safety, and warning requirements of the application are met.

If the equipment is used in any manner that does not conform to the instructions given in this user guide, then the protection provided by the equipment may be impaired.

Change Summary

This volume contains the *ACR Command Language Reference* (formerly *ACR User's Guide, Part 1*). The part number (88-025604-01) remains the same.

The *ACR Parameter and Bit Reference* (formerly *ACR User's Guide, Part 2*) has been broken out into its own volume and now includes the Aries Controller Parameter Reference. The part number (88-025605-01) remains the same.

Both these documents are included in the ACR-View Online Help along with the *ACR Programmer's Guide*.

The following lists the latest additions, changes, and corrections to the *ACR Command Language Reference* and the ACR-View Online Help. Changes to the *ACR Parameter and Bit Reference* (formerly *ACR User's Guide, Part 2*) previous to revision E is also listed in this section.

Revision G Changes

Document 88-025604-01G (*ACR Command Language Reference*) supersedes document 88-025604-01F. Changes associated with this document are notated in this section. Changes previous to this revision are notated on the next several pages.

The most recent changes associated with the *ACR Parameter and Bit Reference* are notated in that volume.

Topic	Description
Introduction	Added to Aries Controller description that PLC commands are not valid; removed references to commutation.
PLC Commands for Aries Controller	Added several other notations throughout this document that PLC commands are not valid for the Aries Controller.
Variable Memory Allocation	Changed reference to "Appendix A" to *ACR Parameter and Bit Reference*.
Example Code Conventions	Added note that parameter and bit examples are usually for Axis0 or Master0.
ADC	Valid command combination ADC SRC added; removed references to non-supported controllers; corrected block diagram for ACR9000 and added block diagram for ACR9030/9040; added Examples 1, 2 and 3.
ADC SRC	Added command.
ASC	Added ASCII Table.
ATTACH AXIS	Removed reference to CMT.
CAM	Refined description of CAM ON TRG and CAM ON TRGP; added Cam Cycle Position and Cycle Number paragraph; added Cam Alignment paragraph.
CAM ON IHPOS	Changed reference from *source encoder counts* to *source counts*; added discussion regarding CAM SRC to Remarks.
CIRCCW	Added Examples 2-4 with drawings.

#DEFINE	Added information regarding changing a DEFINE or assigning a variable, and placement of DEFINE statements; replaced example with new code and discussion.
DIM $V	Remove reference to Chapter 3; replaced example.
DIM LV	Added example.
DRIVE OFF	Added Remarks and Examples.
DRIVE ON	Added Remarks and Examples.
DRIVE RES	Added Remarks and Examples.
DTALK	Removed reference to Example 2; added *Drive Talk for EPL* section.
EPLC ON	Added Example.
FOR/TO/STEP/NEXT	Corrected Format line; added arguments; added example assumptions and Examples 1-4.
GEAR ACC	Added to units: *output units / input unit / input unit* with accompanying Remarks section; expanded remarks for GEAR ACC SLIP.
GEAR DEC	Added to units: *output units / input unit / input unit* with accompanying Remarks section.
GEAR PPU	Expanded code for Example 1; added Example 2.
GEAR SRC	Replaced Example with Examples 1-3.
IF/THEN	Added Remark that this command cannot be placed within IF/ELSE IF/ELSE/ENDIF; added Example 2.
INH	Added optional Timeout argument, and Examples 2 and 4.
INKEY$	Added "THEN" to IF/THEN statements in Example.
INPUT	Corrected Example 1, added Example 2.
INTCAP	Reorganized and consolidated information for entire section.
JOG HOME	Added Remarks, and Examples 2, 3, and 4.
JOG OFF	Added Examples 2 and 3.
LIST	Added Arguments.
OOP	Expanded sections to include ACR9000/9030 for OOP and daughters: OOP BASE, OOP INC, OOP OFF, OOP RND.
PARTNUMBER	New command.
PBOOT	Expanded Remarks; added Examples 2 and 3.
PPU	Specified location of PPU values; rewrote Example 2; added Examples 3-5.
PRINT	Moved Examples 1 and 2 to Examples 2 and 3; added new Example 1.
SLM	Added Remarks; revised Examples 1 and 2.
SPLINE	Corrected code in Example to match drawing.
SPACE$	Corrected spacing of asterisks; expanded Example 1.
WHILE/WEND	Replaced Example with Examples 1 and 2.
Expression Reference	
KBHIT	Split Example 1 and rewrote into Examples 1 and 2.

Revision F Changes

Document 88-025604-01F (*ACR Command Language Reference*) supersedes document 88-025604-01E. Changes associated with this document are notated in this section. Changes previous to this revision are notated on the next several pages.

The most recent changes associated with the *ACR Parameter and Bit Reference* are notated in that volume.

Topic	Description
Introduction	Removed descriptions of non-supported controllers.
Aries Controller	Added to Command and Firmware Release Cross Reference table.
Command Syntax	Parentheses in Arguments and Syntax section: use parentheses if a constant is signed or is changing to a variable.
Example Code Conventions	New section.
INH, **DWL**	Added note of caution for using in non-motion programs.
FOV, **ROV**	Made minor grammatical corrections.

Revision E Changes

Document 88-025604-01E (*ACR Command Language Reference*) supersedes document 88-025604-01D. Changes associated with this document are notated in this section.

The most recent changes associated with the *ACR Parameter and Bit Reference* are notated in that volume. Changes previous to Revision E are notated on the next several pages.

Topic	Description
Added New Commands for ETHERNET Powerlink (EPL)	**DIAG EPLD**, **EPLC**, **EPLC OFF**, **EPLC ON**, **EPLD RES**, **OPEN EPLD**
ATTACH AXIS	Added **EPLD** argument.
Updated and reformatted commands	**BREAK**, **CLEAR**, **CLEAR DPCB**, **CLEAR FIFO**, **CLEAR COM1**, **CLEAR COM2**, **CLEAR STREAM**, **DIAG**, **DIAG EPLD**, **DIAG IP**, **DIAG IP EXEC**, **DIAG IP FSTAT**, **DIAG IP STREAM**, **DIAG XIO**, **DIM**, **DIM $A**, **DIM $V**, **DIM COM1**, **DIM COM2**, **DIM DA**, **DIM DEF**, **DIM DPCB**, **DIM DV**, **DIM FIFO**, **DIM LA**, **DIM LOGGING**, **DIM LV**, **DIM P**, **DIM PLC**, **DIM PROG**, **DIM SA**, **DIM STREAM**, **DIM SV**, **ECHO**, **END**, **ENDP**, **HELP**, **INH**, **MEM**, **PROGRAM**, **REBOOT**, **SRC**, **STREAM**, **VER**
Updated and reformatted all Program Control commands	**AUT**, **BLK**, **HALT**, **LIST**, **LISTEN**, **LRUN**, **NEW**, **PAUSE**, **REM**, **RESUME**, **RUN**, **STEP**, **TROFF**, **TRON**
Command Syntax	Added vertical bar symbol in Arguments and Syntax section.
WHILE/WEND	Corrected group category from Operating System to Program Flow.

Revision D Changes

Documents 88-025604-01D and 88-025605-01D supersede documents 88-025604-01C and 88-025605-01C. Changes associated with these documents and with document clarifications and corrections are as follows:

Topic	Description
Changed titles	Changed title from *ACR User's Guide, Part 1* to *ACR Command Language Reference*. Changed title from *ACR User's Guide, Part 2* to *Parameter and Bit Reference*.
Updated Commands	**ACC**, **CIRCCW**, **CIRCW**, **DEC**, **FOV**, **INT**, **JRK**, **MOV**, **REN**, **RES**, **ROV**, **SINE**, **STP**, **TARC**, **TRJ**, **VEL**
Memory Storage	Added Move Buffers and Aliases to the Memory Storage table under **DIM**.
Binary Host Interface	Moved to ACR Programmer's Guide.
Appendix 3: Output Module Software Configuration Examples	Moved to ACR Programmer's Guide.
OOP INC	Corrected timing references from milliseconds to microseconds.
OPEN DTALK	Corrected mnemonic in format.
CMT	Removed feature from ACR series controller firmware.
SKEW	Removed feature from ACR series controller firmware.
INTCAP-based commands	Added ACR9000 latency information regarding SSI and external inputs.

Revision C Changes

Documents 88-025604-01C and 88-025605-01C supersede documents 88-025604-01B and 88-025605-01B. Changes associated with ACR User Guide revisions, and document clarifications and corrections are as follows:

Topic	Description
Introduction/Getting Started	Added sections on the following: understanding command syntax; Getting Started—Tutorial; Servo Tuning—Tutorial; End-of-Travel Limits; Inverse Kinematics; and IP Addresses, Subnets, and Subnet Masks.
Homing	Added diagrams of homing routines.
Position Velocity Servo Loop	Updated and corrected diagram.
Command Reference	Added the following: **BSC** commands, **DIM** commands, **IP GATE**, **CLEAR STREAM**, **CLEAR DPCB, DIM STREAM**, **STREAM TIMEOUT** Expanded numerous command examples. Corrected or amended syntax format in individual commands to provide more uniform format. Removed the few parameter and bit tables scattered throughout the Command Reference, and added a Related Topics row to the command format block at the beginning of each command.
Parameter Reference	Added description for parameters that have previously lacked any explanation.
Bit Reference	Added description for bits that have previously lacked any explanation.
ECHO	Added note on using command during a program or PLC download.
FLASH	Added note about use of command with battery backed RAM.
GEAR RATIO	Added note about preventing rounding errors.
GOSUB, GOTO	Corrected code example using command to call a loop.
HLBIT	Amended note.
JOG HOMVF	Reference to Home Backup Enable bit corrected.
HOMEVF	Corrected reference to Home Backup Enable bit.
PPU	Expanded command explanation and examples.
PRINT	Added note about command shortcut.
REM	Expanded command explanation and examples.
ROTARY	Expanded command explanation.
SCR	Added notes about encoder argument, and source for homing.

Revision B Changes

Documents 88-025604-01B and 88-025605-01B supersede documents 88-025604-01A and 88-025605-01A. Changes associated with ACR User Guide revisions, and document clarifications and corrections are as follows:

Topic	Description
Hyperlinks	Added links from parameter and flag tables to AcroBASIC commands where appropriate. The links indicate where a command directly controls the parameter or flag.
Stream Flags	Added “Stream XON/XOFF” bit to streams for COM1, COM2, DPCB, and streams 1-5. Allows manual control of XON/XOFF. Added “Stream acknowledge binary write” bit to streams for COM1, COM2, DPCB, and streams 1-5. Bit indicates acknowledgement of binary write over Ethernet.
API Commands & Memory Organization	Updated to current command set.
Command and Firmware Release Cross Reference	Updated to current command set.
Command Reference	Added the following: **ADC FLT**, **BREAK**, **ENC CLOCK**, **ENC DST**, **ENC LIMIT**, **ENC SRC**, **ENC WIDTH**, **FLASH RES**, **OOP INC**, **OOP RND**, **CAM ON IHPOS**, **CAM POFF**
CONFIG IO MODE	Corrected number of modes available for ACR1505.
JOG	Corrected typo in code example.
WHILE/WEND	Corrected typos in code example.
ENC SRC	Corrected descriptions of sources five and six.
FIRMWARE UPGRADE	Corrected procedure for ACR8020.
INTCAP	Added EXPAXIS, ACR1505, ACR8020, and ACR9000 to discussion. Appeared in previous guides, but accidentally dropped from rev A.
LOPASS and NOTCH	Added parameters to Axis Parameters 0-15. See P12288-P149199 and P14366-P16247.

Revision A Changes

Documents 88-025604-01A and 88-025605-01A supersede Online Help (version 1.2.3). Changes associated with ACR User Guide revisions, and document clarifications and corrections are as follows:

Topic	Description
Command Reference	Added the following: Position Maintenance: **PM ACC**, **PM DB**, **PM LIST**, **PM OFF**, **PM ON**, **PM REN**, **PM SCALE**, **PM VEL** Hardware Limits: **HLBIT**, **HLDEC**, **HLIM** Software Limits: **SLDEC**, **SLIM**, **SLM**
Axis Parameters	Added Position Maintenance parameters to Axis Parameter tables
Quaternary Axis Flags (0-15)	Added Ethernet and Position Maintenance flags
Quinary Axis Parameters (0-15)	Added P4600-P4615 Hardware Limit, Software Limit, and Homing Parameters
Quinary Axis Flags (0-15)	Added flags for Hardware Limit, Software Limit, and Homing bits. See Quinary Axis Flags Bit16128-Bit16383 and Bit16384-Bit16639.

Version 1.2.3

Topic	Description
CMT Parameters	Added the Phase Advance bit

Version 1.2.2

Topic	Description
Parameter Reference	Added the following: Streams 1-5 parameters to Flag Parameters Event Counter parameters Stream Parameters to Miscellaneous Parameters
Flag Reference	Added the following: Streams 1-5 Stream Flags for Drive Talk FIFO/Stream 1 Stream Flags Stream 2 Stream Flags DPCB/Stream 3 Stream Flags Stream 4 Stream Flags Stream 5 Stream Flags Connection Status Parameters
Command Reference	Added the **STREAM** command

Version 1.2.1

Topic	Description
Command Reference	Added the following commands: **INVK**, **ENDP**, **PROGRAM**, **IP**, **IP MASK** Deprecated the following commands: **ENC RD ABS**
PLC Reference	Added the following PLC instructions: **CTD**, **LAT**, **TM** Deprecated the following commands: **CNT**, **KR**, **TIM**
Expression Reference	Added the following commands: **ABSF**
Flag Reference	Added documentation for the CANopen feature (ACR9000 only).
Miscellaneous Outputs	Corrected references to PLSn On—moved from Bit96-Bit127 Miscellaneous Outputs to Bit64-Bit95 Miscellaneous Inputs.
Quaternary Axis Flags (0-15)	Corrected index reference in Quaternary Axis Flags (0-7) and Quaternary Axis Flags (8-15). In addition, corrected descriptions for bits 23-28 in same topics.
PLC Flags (8-15)	Corrected bit range. See Bit7168-Bit7423 PLC Flags (8-15) and P4096-P4375 Flag Parameters.
Flag Parameters	Corrected references— P4312-P4319 are Reserved; P4320-P4327 are PLC (8-15). In addition, added P4360-P4375 Quaternary Axis Flags (0-15). See P4096-P4375 Flag Parameters.
CONFIG XAXIS command	Corrected references—applies to ACR8020 and not ACR2000. See the **CONFIG** command.
FOR/TO/STEP/NEXT command	Corrected Format and Group information. See the **FOR/TO/STEP/NEXT** command.
Lineless Program	Added description of Bit5651 to **PROGRAM** and **ENDP** commands; added description of Bit5651 to Miscellaneous Control Group 1 Flags.
CONFIG IO MODE command	Clarified which modes apply to ACR1500 and ACR1505 controllers. See the **CONFIG IO MODE** command.

Introduction

This manual is designed as a reference for the AcroBASIC programming language, used with the ACR series controllers. Use this reference in conjunction with the ACR Programmer's Guide to understand and implement the features of your particular controller. For hardware specific information (such as electrical wiring, connectors, and specifications) please refer to the appropriate ACR hardware installation guide.

ACR Series Controllers

ACR9000: A floating-point DSP-based 8-axis motion controller. It has onboard hardware to read up to eight with the option of ten incremental encoders. It is modular in nature and is offered in 2-, 4-, 6- or 8-axis configurations.

ACR9030: A floating-point DSP-based controller with an ETHERNET Powerlink (EPL) card and two ETHERNET Powerlink connectors. It will support up to 16 axes total. Some or all of the axes may be EPL nodes (that adhere to device profile DSP-402), and a maximum of 8 may be traditional axes. Based on the ACR9000 hardware.

ACR9040: A DC-powered, floating-point DSP-based controller with an ETHERNET Powerlink (EPL) card and two ETHERNET Powerlink connectors. It will support up to 16 EPL nodes that adhere to device profile DSP-402.

ACR1505: A floating-point DSP based 4-axis motion controller, expandable to 12 axes. This board is a PC-AT card only. In the PC-AT bus, the board takes a 32-bit PCI card slot.

ACR8020: A floating-point DSP based 16-axis motion controller. This board will work in standalone mode as well as within a PCI bus chassis.

Aries Controller: A floating-point, DSP-based, single-axis drive/controller based on the Aries drive hardware. It can read up to two encoders, and is configured and programmed within the ACR-View software environment using the AcroBASIC language which combines configuration parameters, status parameters, and text commands. (PLC commands are not valid for the Aries Controller.)

System Reference

This section provides an overview of the executive for the ACR series motion controller. The executive is a pre-emptive, multi-tasking operating system. Depending on your controller, you can have as many as 16 simultaneous tasks open at the same time. Each of these tasks is called a program, and are referenced as PROG0 through PROG15.

- Communication Channels
- Communication Levels
- Multiple Board Communication
- Command Input Modes
- Memory Organization
- API Commands & Memory Organization
- Variable Memory Allocation

Communication Channels

Each controller has communication channels that can be operated simultaneously and attached to various programs. Programs can be running while others are being edited.

All of the channels "wake-up" on power-up when seeing data. Additionally, the serial channels have automatic baud detection that is triggered by receiving one or two carriage returns (ASCII 13) after power-up. For information about the enable/disable jumper settings for automatic baud detection, see the specific hardware installation guide for your controller.

ACR9000: There are six communication channels (or streams) available on the ACR9000 that can be simultaneously open to send and receive data. They are as follows:

COM1: Serial RS-232 or RS-485

STREAM1: USB

STREAM2: Ethernet

STREAM3: Ethernet

STREAM4: Ethernet

STREAM5: Ethernet

ACR2000, ACR8000, and ACR8010: There are three communication channels (or streams) available that can be simultaneously open to send and receive data. The ACR2000 requires the ACRCOMM module. They are as follows:

COM1: Serial RS-232 or RS-422

COM2: Serial RS-232 or RS-422

FIFO: PC/ISA bus, port access

ACR1200: There are two communication channels (or streams) available that can be simultaneously open to send and receive data. They are as follows:

COM1: Serial RS-232 or RS-422

COM2: Serial RS-232 or RS-422

ACR1500: There is one communication channel (or stream) available that can be open to send and receive data. This is as follows:

FIFO: PC/ISA bus, port access

ACR1505 and ACR8020: There are three communication channels (or streams) available that can be simultaneously open to send and receive data. They are as follows:

COM1: Serial RS-232 or RS-422

COM2: Serial RS-232 or RS-422

DPCB: PC/PCI bus, dual port circular buffer

Communication Buffers

As commands are received by a controller, they are stored in an ASCII stream buffer. There is a stream buffer for each of the FIFO, COM1, and COM2 system tasks. The default buffer size is 256 bytes long.

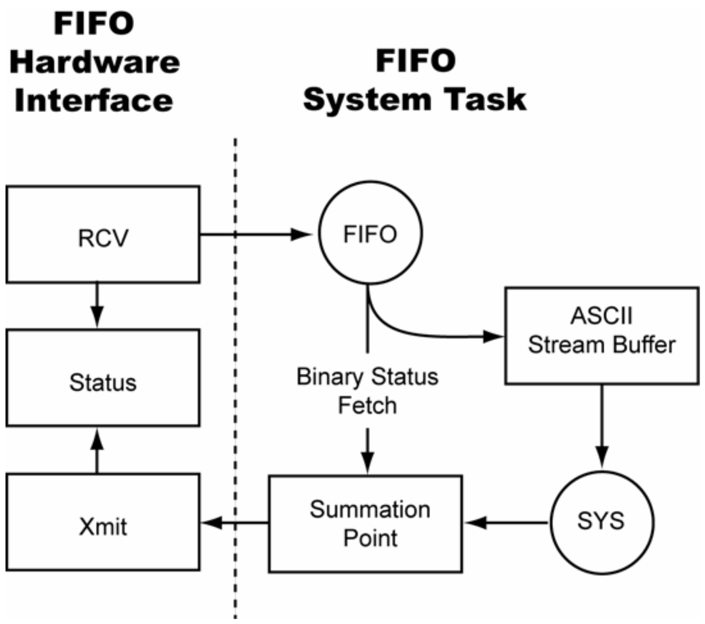
NOTE: FIFO, DPCB, and STREAM are used interchangeably in this discussion.

If the system task cannot process commands faster than the rate at which data comes in—data coming through the controller interface with your PC application—the ASCII buffer can become full. When full, the controller suspends the corresponding system task (FIFO, COM1, or COM2). In turn, the controller's buffer can fill up and cause the PC application to time out based on specific status flags. How an application reacts to timeouts is up to you; refer to the appropriate hardware manual.

The system task for FIFO and COM operates the same. The system task immediately processes all binary commands and sends them back to the controller hardware. Meanwhile, ASCII commands and binary moves are sent to the ASCII stream buffer. After processing an ASCII command or binary move, the result moves to the summation point where it slides between binary command and is sent back to the controller hardware.

You can change the ASCII buffer size (in bytes) with the **DIM** command. ASCII buffer size is limited by the amount of User RAM memory available for dimensioning (see Memory Organization).

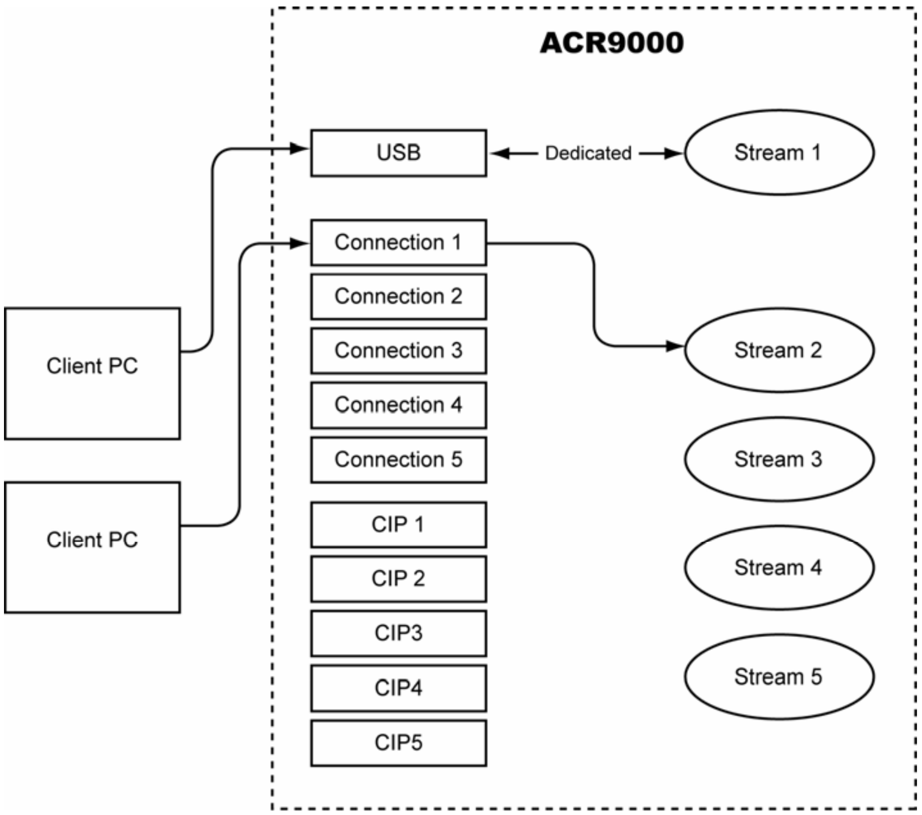
The FIFO system tasks are as follows:



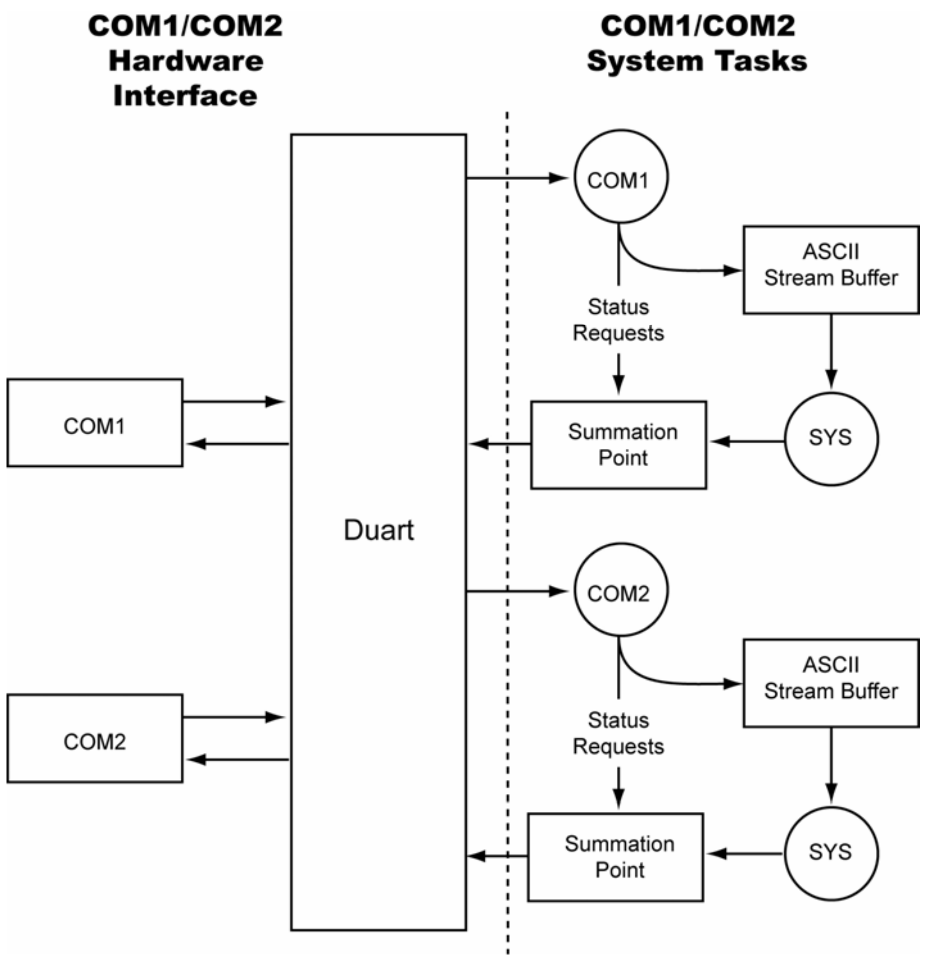
The ACR9000 controller differs—it has COM1, COM2, and STREAM1 through STREAM5. The controller can manage up to five connections, in any combination of ASCII or Ethernet/IP. There are five communication streams for connection, with STREAM1 dedicated to USB.

When making a connection, the controller assigns the connection to the lowest-numbered stream that is available. For example, clients A and B connect to an ACR9000 and are assigned streams two and three, respectively. At some point client A disconnects, freeing stream two. When client C connects to the ACR9000 controller, the controller assigns client C to stream two.

The system task for each STREAM is the same as FIFO. See the FIFO system task diagram above.



The COM1/COM2 system tasks are as follows:



Communication Levels

Communication channels are either at the "system" level or at a "program" level. The command prompt indicates the current level of a communication channel.

Certain commands are limited to a specific level. To determine at which levels a command might be used, refer to the Prompt Level in the command description.

System Level

The "system" level is where a communication channel is after power-up. The command prompt at this level is as follows:

```
SYS>
```

The set of commands you can issue from the system level is limited. You can return to the system level from any other level by issuing the **SYS** command.

Program/PLC Level

The "program" or "PLC" level lets you edit and run individual programs or PLCs (Programmable Logic Controller). The command prompt at the program level is as follows:

```
Pnn>
```

The command prompt at the PLC level is as follows:

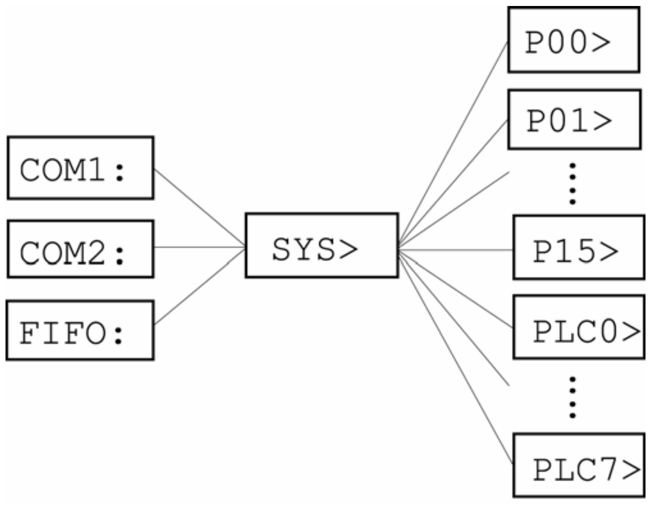
```
PLCn>
```

Where "nn" or "n" represents the currently active program number. To select the program or PLC level from any other level, issue the **PROG** or **PLC** command followed by the program number. For example, the following selects program number 1 no matter which level or program is active:

```
PROG 1
```

To go back to the system level from the program or PLC level, issue the **SYS** command. To move between programs, issue the **PROG** or **PLC** command followed by the desired program or PLC number.

The following shows the various communication channel levels. The communication channels on the left can all be active at the same time and be operating at different levels. For example, "COM1:" could be editing program number 3, while "COM2:" monitors user variables being modified by program number 5.



Multiple Board Communication

PC Bus

Multiple board communication over the PC Bus is handled by using different I/O port addresses and card numbers for each card in the system. Refer to the appropriate hardware manual for Address Selection Switch (SW1) information. Refer to ACR8010 hardware manual for ACR8010 card Plug and Play information.

Up to eight ACR1500, ACR1505, ACR2000, ACR8000, ACR8010, ACR8020 boards can be used in the same system.

Serial Bus

Multiple board communication over the serial ports is handled by using different card numbers for each card in the system, and using the CTRL-A and CTRL-B commands to address the cards. Refer to the appropriate hardware manual for Address Selection Switch (SW1) and Multiple Card Wiring information.

Up to eight ACR1200, ACR1505, ACR2000, ACR8000, ACR8010, ACR8020 board serial ports can be daisy-chained, creating a serial bus.

Symbol	Argument	Function
CTRL-A	# OR <CR>	Turn on card # or all cards
CTRL-B	# OR <CR>	Turn off card # or all cards

The ctrl-A and ctrl-B commands may be terminated by an ASCII character representing the card number, i.e. “0” represents the first card and the zero keys on a keyboard, or by a carriage return (<CR>). A <CR> in this case means “all cards”. Other commands are terminated by a <CR>.

NOTE: Only one board should be enabled (turned on) at a time, to allow for proper handling of unsolicited responses (i.e., error messages, printing tasks, responses from **LRUN** command, etc.) from individual cards.

Example

The following example shows two cards (Card Number 0 and Card Number 1) being controlled via the serial port:

```
Ctrl-B<CR>     : REM Turn all cards off.
Ctrl-A 0<CR>     : REM Turn on Card Number 0.
PROG0<CR>     : REM Program 0 for Card Number 0.
ACC 10<CR>     : REM Set ACC to 10 for Card Number 0.
DEC 10<CR>     : REM Set DEC to 10 for Card Number 0.
VEL 2000<CR>     : REM Set VEL to 2000 for Card Number 0.
Ctrl-B 0<CR>     : REM Turn off Card Number 0.
Ctrl-A 1<CR>     : REM Turn on Card Number 1.
PROG2<CR>     : REM Program 2 for Card Number 1.
ACC 10<CR>     : REM Set ACC to 10 for Card Number 1.
DEC 10<CR>     : REM Set DEC to 10 for Card Number 1.
VEL 1000     : REM Set VEL to 1000 for Card Number 1.
```

System Attachments

The following is an overview of the "objects" involved in system attachment:

Product	Encoders		Command Signals			
	ENCn		DACn		STEPPERn	
	Max	Range (N)	Max	Range (N)	Max	Range (N)
ACR1200	3	0-2	2	0-1	2	0-1
ACR1500	4	0-3	3	0-3	3	0-3
ACR1505	15	0-3 & 10-20	12	0-3 & 10-15		0-3 & 10-15
ACR2000	4	0-3	4	0-3	4	0-3
ACR8000	8	0-7	8	0-7	8	0-7
ACR8010	10	0-9	8	0-7	8	0-7
ACR8020	20	0-19	16	0-15	16	0-15
ACR9000	10	0-9	8	0-7	8	0-7

Product	Attachments					
	AXISn		MASTERn		SLAVEn	
	Max	Range (N)	Max	Range (N)	Max	Range (N)
ACR1200	2	0-1	2	0-1	2	0-1
ACR1500	3	0-3	3	0-3	3	0-3
ACR1505		0-3 & 10-15		0-15		0-15
ACR2000	4	0-3	4	0-3	4	0-3
ACR8000	8	0-7	8	0-7	8	0-7
ACR8010	8	0-7	8	0-7	8	0-7
ACR8020	16	0-15	16	0-15	16	0-15
ACR9000	8	0-7	8	0-7	8	0-7

NOTE: In order for an axis to do a motion profile in either MDI (Manual Data Input) or program mode, it must be attached as the slave of a master and the master must be attached to a program.

Axes are accessed with axis names of up to 4 characters in length. This name is assigned to the axis when it is attached to its master. See the ATTACH command for examples.

The question of how many masters to use and what axes to attach to it is largely a user choice made initially on the type of machine. If there are 6 axes in total and they are broken into two XYZ pick and place robots, then use two masters and two programs, attaching three axes to each of the masters. Since the axes are attached to different masters, they can be named X, Y and Z in both programs.

Once the relationship between a program, master, and axes has been established, it can be canceled by using the **DETACH** command.

Command Input Modes

Most commands can be executed either in MDI (Manual Data Input) mode or program mode (from within a stored program.) Some commands can only be issued as MDI commands and others can only appear in stored programs.

In MDI mode, the commands are executed immediately as they are entered while in the program mode, the commands are stored in memory as they are entered. They can be executed later by issuing a **RUN** command in the MDI mode. Program mode entry is not allowed from the system level.

Any command that is preceded by a valid line number will get stored into the memory of the currently selected program (program mode). Valid line numbers are in the range of 1 to65000. Each line in a program must have a line number. These line numbers are also used as destination addresses for **GOTO** commands.

As an example, the following set of commands will do the identical motion profile but will illustrate the difference between the MDI mode and the program mode.

Program Example

```
PROG0
ACC 10000 DEC 10000 STP10000 VEL 20000
X100000
END
RUN
```

MDI Example

```
PROG0
ACC10000 DEC 10000 STP 10000 VEL 20000
X100000
```

Memory Organization

There are five types of memory blocks in the ACR1200, ACR1500, ACR2000, ACR8000, ACR8010 memory organization as described below:

EPROM

The EPROM is Electrically Programmable Read Only Memory. The EPROM's main function is to store the executable firmware code. The EPROM is programmed at the factory and is not programmable by the user.

The EPROM-based code in the ACR8000 runs at one wait-state (148 nanoseconds using a 27 MHz System Clock), at two wait-states for the ACR1200 and ACR1500 (150ns using a 40 MHz System Clock), and at two wait-states for the Standard Memory ACR2000 (120 nanoseconds using a 50 MHz System Clock). The code for the ACR8010 and the Expanded Memory ACR2000 does not run out of the EPROM, but out of the system RAM (see below).

The ACR8000, ACR1200, ACR1500, and Standard Memory ACR2000 EPROMs also contain a section of shadow code, which is loaded into the System RAM memory at power-up or reset. The ACR8010 and the Expanded Memory ACR2000 EPROM load all of the executable firmware code into the Expanded System RAM memory at power-up or reset. The code that is loaded into RAM runs at zero wait-states.

System RAM

The System RAM is Static Random Access Memory. This memory is not battery backed-up.

The System RAM's functions are to store system parameters and flag information and to store the shadow code, which is copied into the System RAM from the EPROM by the DSP.

The System RAM runs at zero wait-state for the ACR8000 (74 nanoseconds using a 27 MHz System Clock), the ACR1200 and ACR1500 (50 nanoseconds using a 40 MHz System Clock), and Standard Memory ACR2000 (40 nanoseconds using a 50 MHz System Clock). This allows fast access by the DSP for the system parameters and flags information, as well as the speed critical shadow code for the main servo system interrupt.

The executable code for the ACR8010 and the Expanded Memory ACR2000 is copied into the System RAM from the EPROM by the DSP. All of the executable code then runs at zero wait-state (40 nanoseconds using a 50 MHz System Clock for Expanded Memory ACR2000, 33 nanoseconds using a 60 MHz System Clock for ACR8010), which increases system performance.

User RAM

The User RAM is battery backed-up Static Random Access Memory for the ACR1200, ACR8000, and the ACR8010 board. This memory is battery backed-up for ACR2000 boards with the ACRCOMM Module option.

The main function of the User Ram is to store user information. The user determines what is stored into these memories by using the **DIM** command. User programs, PLC programs, FIFO, COM1, and COM2 stream buffers, **CAM** and ballscrew tables, global variables, program variables, and program arrays are all stored in these memories.

There is also dedicated section of User RAM that stores Global Data. The global data provides User RAM locations and dimensioning information, which is used by the DSP, for all of the above user data.

This memory is not battery backed-up for ACR1500 boards or ACR2000 boards without the ACRCOMM Module option. The user must load the data each time the board is powered-up.

EEPROM

The EEPROM is an Electrically Erasable Programmable Read-Only Memory on the ACR8000 board.

The EEPROM's functions include storing system parameters and board configuration information.

System parameters are stored using the **ESAVE** command and loaded into the card using the **ELOAD** command. System parameters can be updated in the EEPROM by first erasing the existing parameters, using the **ERASE** command, and then using the **ESAVE** command to store the new information.

The system parameters stored in the EEPROM using the **ESAVE** command include system attachments, master parameters (**ACC**, **DEC**, and **STP** ramps, and **VEL**, **FVEL**, and **IVEL** values), axis parameters (gain and limit setting, **PPU** and **VECDEF** values, and ON/OFF states), encoder multipliers, **DAC** gains and offsets, and **ADC** gains and offsets.

Board configuration information is stored when the **CONFIG** commands are used.

Flash

The Flash is an Electrically Erasable Programmable Read-Only Memory on the ACR1200, ACR1500, ACR2000, and ACR8010 boards.

The Flash's functions include storing system parameters, board configuration information and user programs.

System parameters are stored using the **ESAVE** command and loaded into the card using the **ELOAD** command. System parameters can be updated in the Flash by first erasing the existing parameters, using the **ERASE** command, and then using the **ESAVE** command to store the new information.

The system parameters stored in the Flash using the **ESAVE** command include system attachments, master parameters (**ACC**, **DEC**, and **STP** ramps, and **VEL**, **FVEL**, and **IVEL** values), axis parameters (gain and limit setting, **PPU** and **VECDEF** values, and ON/OFF states), encoder multipliers, **DAC** gains and offsets, and **ADC** gains and offsets.

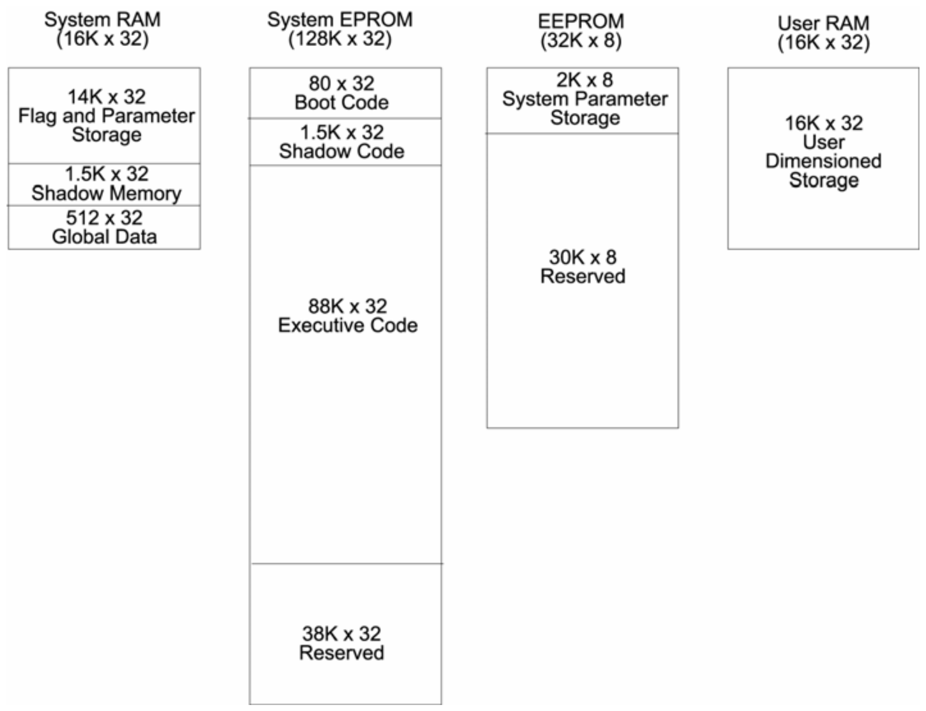
Board configuration information is stored when the **CONFIG** commands are used.

Program storage uses the **FLASH** commands (**FLASH LOAD**, **FLASH ERASE**, **FLASH SAVE**, **FLASH IMAGE**). At power-up or reset, the DSP detects if programs are present in the Flash, and if they are present, loads them into User RAM, overwriting any battery back-up programs. The tables, buffers, variables and arrays stored in the User RAM are not written over.

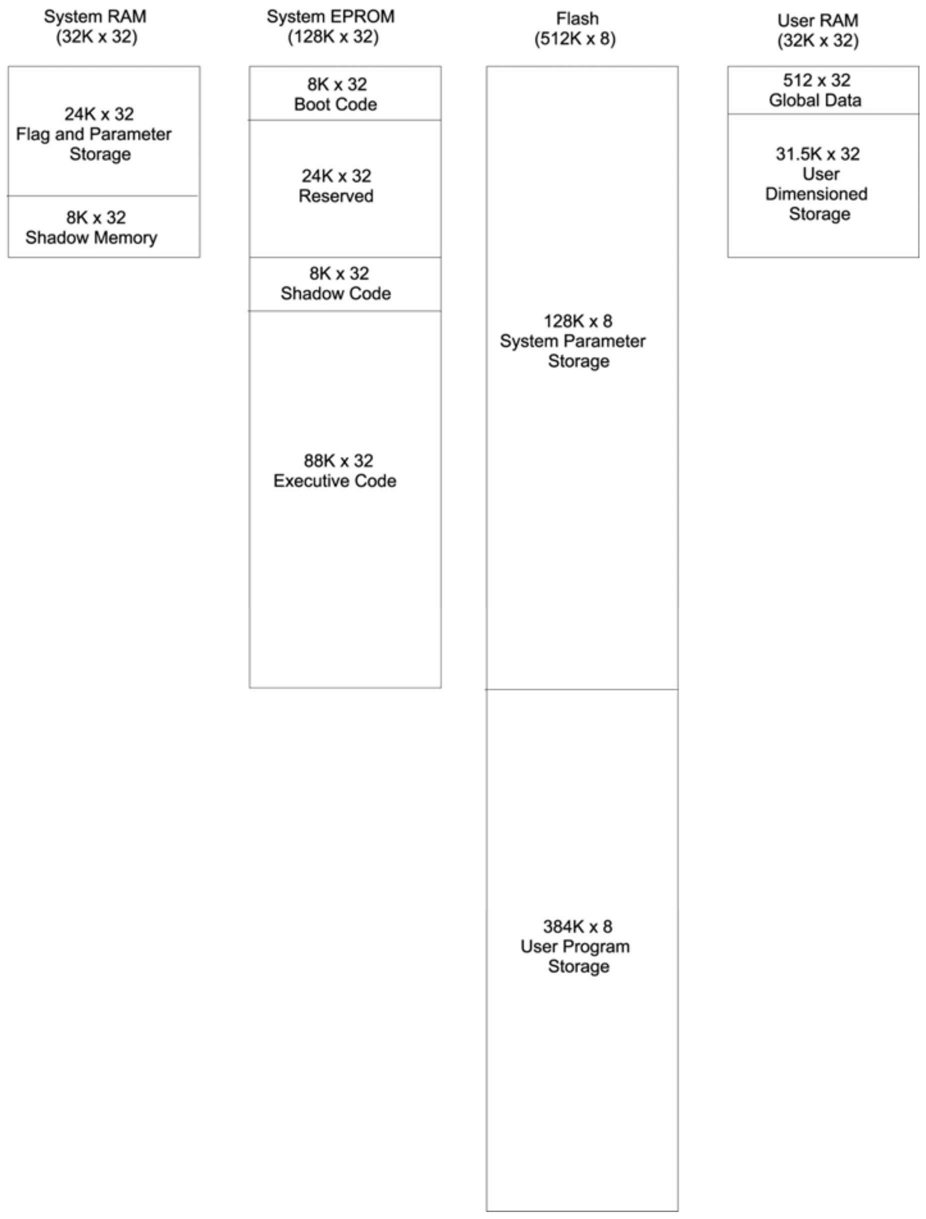
Board Memory Organization

The following drawings illustrate the memory organization of the ACR1200, ACR1500, ACR1505, ACR2000, ACR8000, ACR8010, ACR8010 Expanded, ACR8020, and ACR9000 boards:

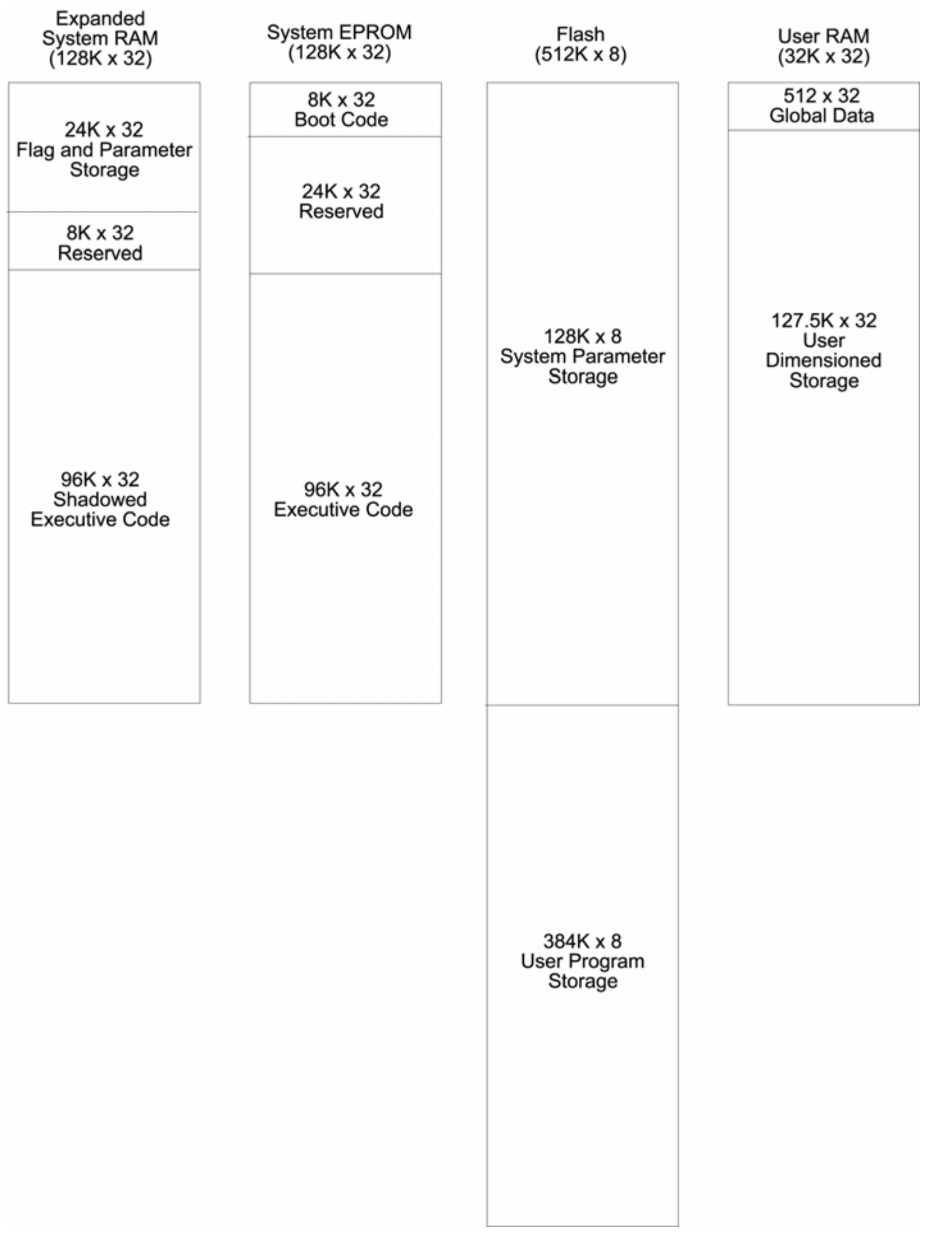
ACR8000


System RAM
(16K x 32)
14K x 32
Flag and Parameter
Storage
1.5K x 32
Shadow Memory
512 x 32
Global Data
System EPROM
(128K x 32)
80 x 32
Boot Code
1.5K x 32
Shadow Code
88K x 32
Executive Code
38K x 32
Reserved
EEPROM
(32K x 8)
2K x 8
System Parameter
Storage
30K x 8
Reserved
User RAM
(16K x 32)
16K x 32
User
Dimensioned
Storage

ACR1200, ACR1500, and ACR2000 (128K)



ACR8010, ACR8010 Expanded, ACR8020, ACR1505, ACR9000, and ACR2000 (512KB)


Expanded System RAM (128K x 32)
24K x 32 Flag and Parameter Storage
8K x 32 Reserved
96K x 32 Shadowed Executive Code
System EPROM (128K x 32)
8K x 32 Boot Code
24K x 32 Reserved
96K x 32 Executive Code
Flash (512K x 8)
128K x 8 System Parameter Storage
384K x 8 User Program Storage
User RAM (32K x 32)
512 x 32 Global Data
127.5K x 32 User Dimensioned Storage

API Commands & Memory Organization

The following table shows the type(s) of memory associated with system commands:

Command	System RAM	User RAM	EPROM	EEPROM/ Flash	N/A
ACC	X			X	
ADC	X			X	
ADCX	X			X	
ALM	X			X	
ATTACH	X			X	
AUT					X
AXIS	X			X	
BKL	X				
BLK					X
BLM	X				
BRESET		X			
BSC		X			
CAM		X			
CIRCCW	X				
CIRCW	X				
CLEAR		X			
CLOSE	X				
CLR	X				
CONFIG				X	
CPU					X
DAC	X			X	
DEC	X			X	
DEF	X	X		X	
#DEFINE				X	
DETACH	X			X	
DGAIN	X			X	
DIAG					X
DIM		X			
DIN	X				
DIP	X				
DRIVE	X				
DTALK	X				
DWIDTH	X			X	

Command	System RAM	User RAM	EPROM	EEPROM/ Flash	N/A
DWL					X
DZL	X				
DZU	X				
ECHO					X
ELOAD				X	
ENC	X			X	
END					X
ENDP					X
EPLC					X
ERASE				X	
ESAVE				X	
EXC	X				
F	X				
FBVEL	X			X	
FFACC	X			X	
FFVEL	X			X	
FFVC	X				
FLASH				X	
FLT	X				
FLZ	X				
FIRMWARE					X
FOR/TO/STEP/NEXT					X
FOV	X				
FVEL	X			X	
FSTAT	X				
GEAR		X			
GOSUB					X
GOTO					X
HALT					X
HDW	X				
HELP					X
HLBIT	X			X	
HLDEC	X				
HLIM	X				
HSINT	X				
IDELAY	X			X	
IF/ELSE					X

Command	System RAM	User RAM	EPROM	EEPROM/ Flash	N/A
IF/ELSE/ENDIF					
IF/THEN					X
IGAIN	X			X	
IHPOS	X				
ILIMIT	X			X	
INH					X
INPUT					X
INT	X				
INTCAP	X				
INVK	X				
IP	X			X	
IPB	X			X	
ITB	X			X	
IVEL	X			X	
JLM	X			X	
JOG	X			X	
JRK	X				
KVF	X				
KVI	X				
KVP	X				
LIMIT	X				
LIST					X
LISTEN					X
LOCK					X
LOOK	X				
LOPASS	X				
LRUN					X
MASK	X				
MASTER	X			X	
MAXVEL	X				
MBUF		X			
MEM		X			
MODE					X
MOV	X				
MSEEK	X				
MULT	X			X	
NEW		X			

Command	System RAM	User RAM	EPROM	EEPROM/ Flash	N/A
NORM	X				
NOTCH	X				
NURB	X				
OFFSET	X				
OOP	X			X	
OPEN					X
PASSWORD	X				
PAUSE					X
PBOOT		X			
PERIOD	X				
PGAIN	X			X	
PLC		X			
PLS	X				
PM	X			X	
PPU	X			X	
PRINT					X
PROG		X			
PROGRAM					X
PROM		X	X		
RATCH	X				
REBOOT					X
REM					X
REN	X				
RES	X				
RESUME					X
RETURN					X
ROTARY	X				
ROTATE	X				
ROV	X				
RUN					X
SAMP		X			
SCALE	X				
SET	X				
SINE	X				
SLDEC	X				
SLIM	X				
SLM	X				

Command	System RAM	User RAM	EPROM	EEPROM/ Flash	N/A
SPLINE	X				
SRC	X				
STREAM					X
STEP					X
STP	X			X	
SYNC	X		X		
SYS					X
TALK TO	X				
TANG	X				
TARC	X				
TLM	X			X	
TMOV	X				
TOV	X				
TRG	X				
TRJ	X				
TROFF					X
TRON					X
UNLOCK					X
VECDEF	X			X	
VECTOR	X				
VEL	X			X	
VER		X	X		
WHILE/WEND					X

Variable Memory Allocation

For the following definitions, xxx and yyy are positive numbers. Their range depends on memory limitations. There are two sets of memory types accessible by each program, global system and local user parameters.

Each parameter number is preceded by the letter "P". For example, the following command will load system parameter 4097 with the value 123:

```
P4097 = 123
```

In addition to the global system parameters that can be accessed by all programs, each program can dimension (allocate) local parameters. These parameters can be either single variables or arrays of variables.

The following formats are used to access the different types of variables:

- `Pxxx:` Global system variable xxx.
- `BITxxx:` Global bit flag xxx
- `LVxxx:` Local Long (32 bit integer) variable xxx.
- `LAxxx(yyy):` Local Long array number xxx and index yyy.
- `SVxxx:` Local Single (32 bit floating point) variable xxx.
- `SAxxx(yyy):` Local Single array number xxx and index yyy.
- `DVxxx:` Local Double (64 bit floating point) variable xxx.
- `DAxxx(yyy):` Local Double array number xxx and index yyy.
- `$Vxxx:` Local String (packed 8 bit) variable xxx.
- `$Axxx(yyy):` Local String array number xxx and index yyy.

Global user variables are referenced as P0-P4095. The actual number of global user variables is determined by the **DIM P** command from the system level. Global system parameters are referenced according to the tables in the *ACR Parameter and Bit Reference*.

Local user variables are referenced by their type, followed by a number. The following command will load the number 1234 into the first long integer variable:

```
LV0 = 1234
```

The local arrays are referenced by an array number and index. The following command will load the number 1234 into the fifth element of the first long integer array:

```
LA0(4) = 1234
```

Only the amount of memory remaining limits how many parameters can be allocated. The **DIM** command is used to declare the parameters and the **CLEAR** command to release them. Full floating point math is allowed to operate on global and local variables.

Parametric Evaluation

A built-in floating-point evaluator operates on the global and local variables. The following operators are available:

- Arithmetic: **+**, **-**, *****, **/**, ******
- Trigonometric: **SIN**, **COS**, **TAN**, **ATAN**, **ACOS**, **ASIN**
- Hyperbolic: **SINH**, **COSH**, **TANH**, **ATANH ACOSH**, **ASINH**
- Logarithmic: **LOG**, **LN**
- Comparison: **=**, **<>**, **<**, **>**, **>=**, **<=**
- Logical: **AND**, **OR**, **XOR**, **NAND**, **NOR**, **XNOR**, **NOT**, **<<**, **>>**
- Miscellaneous: **SQRT**, **RND**
- String: **CHR$**, **ASC**, **LEN**, **STR$**, **VAL**, **INSTR**, **LCASE$**, **UCASE$**, **SPACE$**, **STRING$**, **LEFT$**, **RIGHT$**, **MID$**, **INKEY$**, **KBHIT**

The following are an example of valid statements assuming that the parameters have been properly dimensioned and the X and Y attachments have been defined:

```
LV1 = LV2+LV3*(LV5+LV6)

IF ((LV5+LV6*LV9) < 10) THEN GOTO 100

X(LV1+LV2) Y(3*LV5)
```

Note that the arguments of X and Y are enclosed in parentheses. Parametric arguments are given this way to a command. All commands will accept parametric arguments.

A slash mark (/) indicates an incremental distance for axis moves. It must precede the numerical or parametric argument for incremental axis moves. For example, the following command moves the X-axis 20 units in the positive direction from its current location:

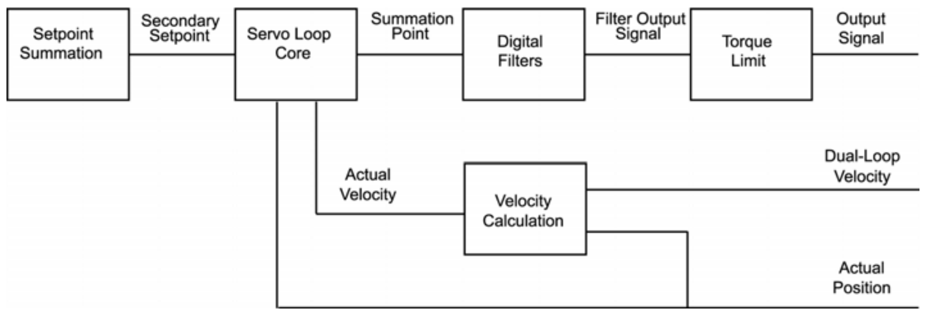
```
X /20
```

It is possible to mix absolute, incremental, and parametric moves as follows:

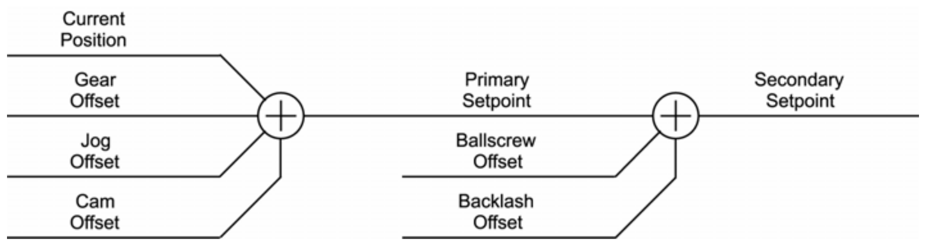
```
X2.5 Y/1.23 Z/(DV2*DV3)
```

Servo Loop Illustrations

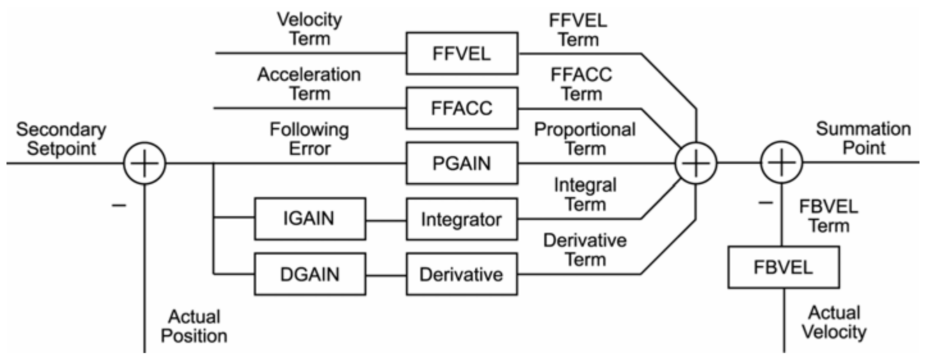
Servo Loop



Setpoint Summation

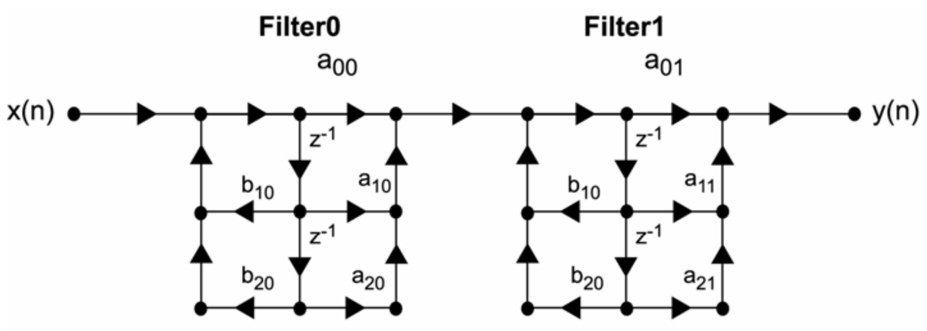


Servo Loop Core



Digital Filters

Filter Equations



$$H(z) = \left(\frac{a_{00} + a_{10}z^{-1} + a_{20}z^{-2}}{1 - b_{10}z^{-1} - b_{20}z^{-2}} \right) \left(\frac{a_{01} + a_{11}z^{-1} + a_{21}z^{-2}}{1 - b_{11}z^{-1} - b_{21}z^{-2}} \right)$$

Filter Parameters

Filter 0	Axis Number							
	0	1	2	3	4	5	6	7
b2 Coefficient	12336	12592	12848	13104	13360	13616	13872	14128
a2 Coefficient	12337	12593	12849	13105	13361	13617	13873	14129
b1 Coefficient	12338	12594	12850	13106	13362	13618	13874	14130
a1 Coefficient	12339	12595	12851	13107	13363	13619	13875	14131
a0 Coefficient	12340	12596	12852	13108	13364	13620	13876	14132

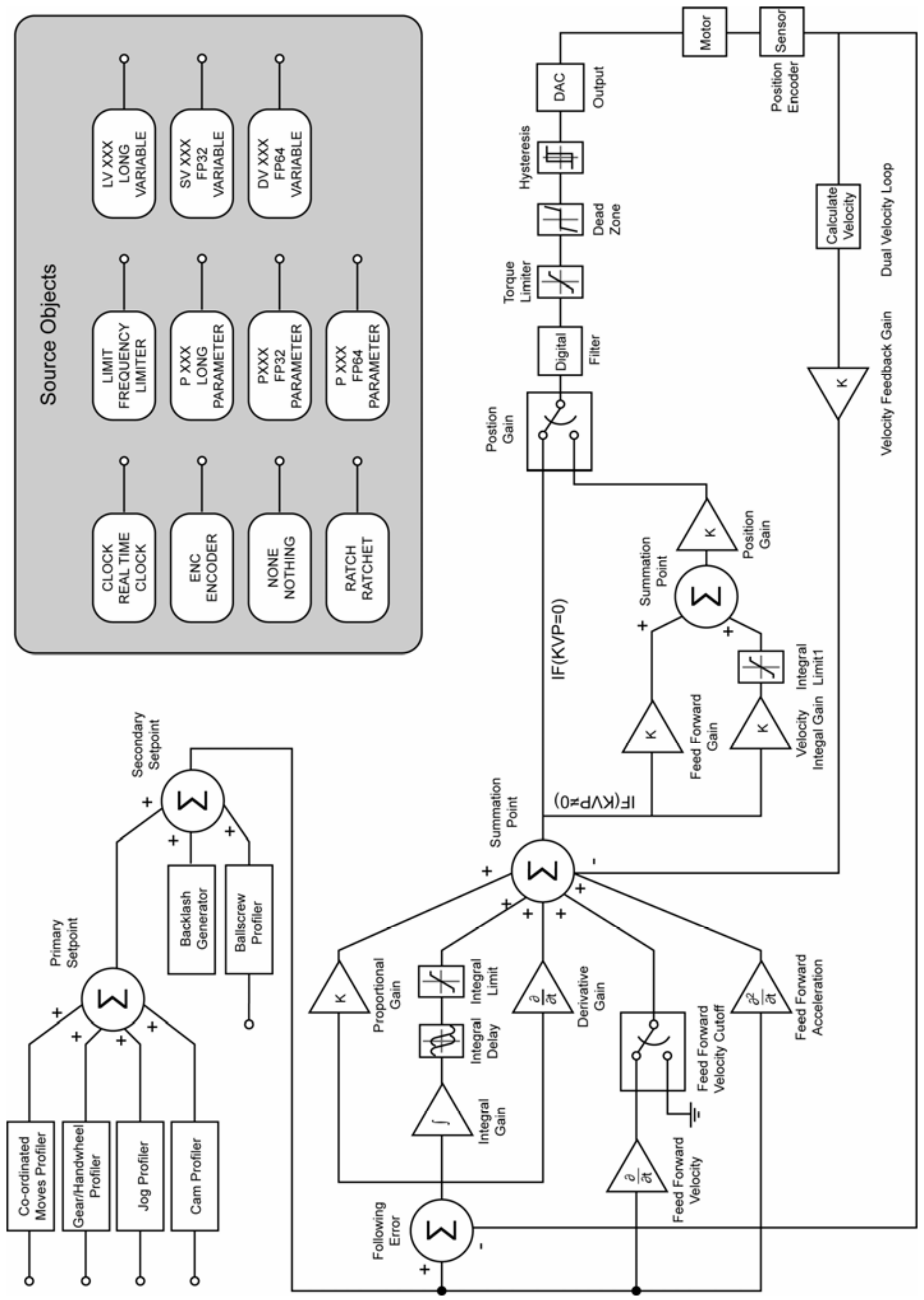
Filter 1	Axis Number							
	0	1	2	3	4	5	6	7
b2 Coefficient	12341	12597	12853	13109	13365	13621	13877	14133
a2 Coefficient	12342	12598	12854	13110	13366	13622	13878	14134
b1 Coefficient	12343	12599	12855	13111	13367	13623	13879	14135
a1 Coefficient	12344	12600	12856	13112	13368	13624	13880	14136
a0 Coefficient	12345	12601	12857	13113	13369	13625	13881	14137

Filter Flags

Control	Axis Number							
	0	1	2	3	4	5	6	7
Filter Activate	786	818	850	882	914	946	978	1010

Position Velocity Servo Loop

The following illustrates a servo loop with dead band and position velocity loop:



Command Reference

The AcroBASIC programming language gives you the building blocks to create superior motion control. The language comprises simple ASCII mnemonic commands, and uses a parent daughter approach. The parent statement can have one or more daughter statements, and the daughter statement is viewed as a sub-statement of the parent.

For more information about command formats, arguments, and syntax, see Command Syntax.

Command and Firmware Release Cross Reference

The following table shows the commands available for each of the ACR Motion Controller family of boards. This table also indicates at what firmware level a command has been added above the base firmware level 1.13.03, as well as what firmware level that the boards have been added.

Command	**ACR1505 (1.18.09)**	**ACR8020 (1.18.06 Upd09)**	**ACR9000 (1.19.0)**	**ACR9030 (1.19.0)**	**ACR9040 (1.19.0)**	**Aries Controller**
ACC	Y	Y	Y	Y	Y	Y
ADC	—	—	—	—	—	—
FLT	N	N	1.18.16	N	N	N
GAIN	Y	Y	Y	N	N	N
MODE	Y	Y	Y	N	N	N
MAX	Y	Y	Y	N	N	N
NEG	Y	Y	Y	N	N	N
OFFSET	Y	Y	Y	N	N	N
ON	Y	Y	Y	N	N	N
OFF	Y	Y	Y	N	N	N
POS	Y	Y	Y	N	N	N
SCALE	Y	Y	Y	N	N	N
ADCX	Y	Y	N	N	N	N
MODE	Y	Y	N	N	N	N
MAX	Y	Y	N	N	N	N
ALM	Y	Y	Y	Y	Y	Y
ATTACH	Y	Y	Y	Y	Y	Y
AXIS	Y	Y	Y	Y	Y	Y
MASTER	Y	Y	Y	Y	Y	Y
SLAVE	Y	Y	Y	Y	Y	Y
AUT	Y	Y	Y	Y	Y	Y
AXIS	Y	Y	Y	Y	Y	Y
OFF	Y	Y	Y	Y	Y	Y
ON	Y	Y	Y	Y	Y	Y
BKL	Y	Y	Y	Y	Y	Y
BLK	Y	Y	Y	Y	Y	Y
BLM	Y	Y	Y	Y	Y	Y
BRESET	N	Y	Y	Y	Y	Y
BSC	Y	Y	Y	Y	Y	Y
DIM	Y	Y	Y	Y	Y	Y
FLZ	Y	Y	Y	Y	Y	Y
OFF	Y	Y	Y	Y	Y	Y
OFFSET	Y	Y	Y	Y	Y	Y
ON	Y	Y	Y	Y	Y	Y
RES	Y	Y	Y	Y	Y	Y

Command	ACR1505 (1.18.09)	ACR8020 (1.18.06 Upd09)	ACR9000 (1.19.0)	ACR9030 (1.19.0)	ACR9040 (1.19.0)	Aries Controller
SCALE	Y	Y	Y	Y	Y	Y
SEG	Y	Y	Y	Y	Y	Y
SHIFT	Y	Y	Y	Y	Y	Y
SRC	Y	Y	Y	Y	Y	Y
CAM	—	—	—	—	—	—
CLEAR	1.18.16	1.18.16	1.18.16	Y	Y	Y
DIM	Y	Y	Y	Y	Y	Y
FLZ	Y	Y	Y	Y	Y	Y
OFF	Y	Y	Y	Y	Y	Y
OFFSET	Y	Y	Y	Y	Y	Y
ON	Y	Y	Y	Y	Y	Y
ON TRG	Y	Y	Y	Y	Y	Y
ON TRGP	Y	Y	Y	Y	Y	Y
ON IHPOS	Y	1.18.08 Upd 13	Y	Y	Y	Y
POFF	Y	1.18.08 Upd 13	Y	Y	Y	Y
RES	Y	Y	Y	Y	Y	Y
SCALE	Y	Y	Y	Y	Y	Y
SHIFT	Y	Y	Y	Y	Y	Y
SEG	Y	Y	Y	Y	Y	Y
SRC	Y	Y	Y	Y	Y	Y
SRC RES	Y	Y	Y	Y	Y	Y
CIRCCW	Y	Y	Y	Y	Y	Y
CIRCW	Y	Y	Y	Y	Y	Y
CLEAR	Y	Y	Y	Y	Y	Y
CLOSE	Y	Y	Y	Y	Y	Y
CLR	Y	Y	Y	Y	Y	Y
CONFIG	—	—	—	—	—	—
CLEAR	Y	Y	Y	Y	Y	Y
IO	Y	Y	Y	Y	Y	Y
IO INPUT	Y	N	N	N	N	N
IO MODE	Y	N	N	N	N	N
IO OUTPUT	Y	N	N	N	N	N
XAXIS	Y	1.18.07	N	N	N	N
XIO	Y	Y	Y	Y	Y	N
CPU	Y	Y	Y	Y	Y	Y
DAC	Y	Y	Y	Y	Y	N
DEC	Y	Y	Y	Y	Y	Y
#DEFINE	Y	1.18.07	Y	Y	Y	Y
DETACH	Y	Y	Y	Y	Y	Y
DGAIN	Y	Y	Y	Y	Y	Y
DIAG	Y	Y	Y	Y	Y	Y
EPLD	N	N	N	Y	Y	N
IP	N	N	1.18.12	Y	Y	Y
IP EXEC	N	N	1.18.12	Y	Y	N

Command	ACR1505 (1.18.09)	ACR8020 (1.18.06 Upd09)	ACR9000 (1.19.0)	ACR9030 (1.19.0)	ACR9040 (1.19.0)	Aries Controller
IP FSTAT	N	N	1.18.12	Y	Y	Y
IP STREAM	N	N	1.18.12	Y	Y	Y
XIO	N	N	1.18.12	Y	Y	N
DIM	—	—	—	—	—	—
PROG	Y	Y	Y	Y	Y	Y
PLC	Y	Y	Y	Y	Y	N
P	Y	Y	Y	Y	Y	Y
FIFO	Y	Y	Y	Y	Y	N
COM1	Y	Y	Y	Y	Y	N
COM2	Y	Y	Y	Y	Y	N
LOGGING	Y	Y	Y	Y	Y	Y
MBUF	Y	Y	Y	Y	Y	Y
DIN	Y	Y	Y	Y	Y	Y
DIP	Y	Y	Y	Y	Y	Y
DRIVE	N	N	Y	Y	Y	Y
OFF	N	N	Y	Y	Y	Y
ON	N	N	Y	Y	Y	Y
RES	N	N	Y	Y	Y	Y
DTALK	N	N	Y	Y	Y	N
DWIDTH	Y	Y	Y	Y	Y	Y
DWL	Y	Y	Y	Y	Y	Y
DZL	Y	Y	Y	Y	Y	Y
DZU	Y	Y	Y	Y	Y	Y
ECHO	Y	Y	Y	Y	Y	Y
ELOAD	Y	Y	Y	Y	Y	N
ENC	Y	Y	Y	Y	Y	N
CLOCK	N	N	1.18.15	Y	Y	N
DST	N	N	1.18.15	Y	Y	N
LIMIT	N	N	1.18.15	Y	Y	N
MULT	Y	Y	Y	Y	Y	N
SRC	N	N	Y	Y	Y	N
WIDTH	N	N	1.18.15	Y	Y	N
END	Y	Y	Y	Y	Y	Y
ENDP	Y	1.18.07	Y	Y	Y	Y
EPLC	N	N	N	Y	Y	N
OFF	N	N	N	Y	Y	N
ON	N	N	N	Y	Y	N
ERASE	Y	Y	Y	Y	Y	Y
ESAVE	Y	Y	Y	Y	Y	N
EXC	Y	Y	Y	Y	Y	Y
F	Y	Y	Y	Y	Y	Y
FBVEL	Y	Y	Y	Y	Y	Y

Command	ACR1505 (1.18.09)	ACR8020 (1.18.06 Upd09)	ACR9000 (1.19.0)	ACR9030 (1.19.0)	ACR9040 (1.19.0)	Aries Controller
FFACC	Y	Y	Y	Y	Y	Y
FFVC	Y	Y	Y	Y	Y	Y
FFVEL	Y	Y	Y	Y	Y	Y
FIRMWARE	Y	1.18.06 Upd 14	Y	Y	Y	N
BACKUP	Y	1.18.06 Upd 14	N	N	N	N
CHECKSUM	Y	1.18.06 Upd 14	N	N	N	N
UPGRADE	Y	1.18.06 Upd 14	Y	Y	Y	N
FLASH	—	—	—	—	—	Y
ERASE	Y	Y	Y	Y	Y	Y
LOAD	Y	Y	Y	Y	Y	Y
IMAGE	Y	Y	Y	Y	Y	N
RES	Y	1.18.10	Y	Y	Y	Y
SAVE	Y	Y	Y	Y	Y	Y
FLT	Y	Y	Y	Y	Y	Y
OFF	Y	Y	Y	Y	Y	Y
ON	Y	Y	Y	Y	Y	Y
OUT	Y	Y	Y	Y	Y	Y
SRC	Y	Y	Y	Y	Y	Y
FLZ	Y	Y	Y	Y	Y	Y
FOR/TO /STEP/NEXT	Y	1.18.07	Y	Y	Y	Y
FOV	Y	Y	Y	Y	Y	Y
FVEL	Y	Y	Y	Y	Y	Y
FSTAT	1.18.06 Upd 9	1.18.06 Upd 9	1.18.12	Y	Y	Y
CLEAR	1.18.06 Upd 9	1.18.06 Upd 9	1.18.12	Y	Y	Y
OFF	1.18.06 Upd 9	1.18.06 Upd 9	1.18.12	Y	Y	Y
ON	1.18.06 Upd 9	1.18.06 Upd 9	1.18.12	Y	Y	Y
PERIOD	1.18.06 Upd 9	1.18.06 Upd 9	1.18.12	Y	Y	Y
GEAR	Y	Y	Y	Y	Y	Y
ACC	Y	Y	Y	Y	Y	Y
CLEAR	Y	Y	Y	Y	Y	Y
DEC	Y	Y	Y	Y	Y	Y
IHPOS	Y	Y	Y	Y	Y	Y
MAN	Y	Y	Y	Y	Y	Y
MIN	Y	Y	Y	Y	Y	Y
OFF	Y	Y	Y	Y	Y	Y
ON	Y	Y	Y	Y	Y	Y
PPU	Y	Y	Y	Y	Y	Y
RATIO	Y	Y	Y	Y	Y	Y
RES	Y	Y	Y	Y	Y	Y
SRC	Y	1.18.07	Y	Y	Y	Y
TRG	Y	1.18.07	Y	Y	Y	Y
TRGP	1.18.18	1.18.18	1.18.18	Y	Y	Y
GOSUB	Y	Y	Y	Y	Y	Y

Command	ACR1505 (1.18.09)	ACR8020 (1.18.06 Upd09)	ACR9000 (1.19.0)	ACR9030 (1.19.0)	ACR9040 (1.19.0)	Aries Controller
GOTO	Y	Y	Y	Y	Y	Y
HALT	Y	Y	Y	Y	Y	Y
HDW	Y	Y	Y	Y	Y	Y
HELP	Y	Y	Y	Y	Y	Y
HLBIT	N	N	1.18.15	Y	Y	Y
HLDEC	N	N	1.18.15	Y	Y	Y
HLIM	N	N	1.18.15	Y	Y	Y
HSINT	Y	Y	Y	Y	Y	Y
IDELAY	Y	Y	Y	Y	Y	Y
IF/ELSE IF/ELSE/ENDIF	Y	1.18.07	Y	Y	Y	Y
IF/THEN	Y	Y	Y	Y	Y	Y
IGAIN	Y	Y	Y	Y	Y	Y
IHPOS	Y	Y	Y	Y	Y	Y
ILIMIT	Y	Y	Y	Y	Y	Y
INH	Y	Y	Y	Y	Y	Y
INPUT	Y	Y	Y	Y	Y	Y
INT	Y	Y	Y	Y	Y	Y
INTCAP	Y	Y	Y	Y	Y	Y
OFF	Y	Y	Y	Y	Y	Y
Fixed Registers	Y	Y	Y	Y	Y	Y
Register Select	Y	Y	Y	Y	Y	Y
INVK	1.18.12	1.18.12	1.18.12	Y	Y	Y
OFF	1.18.12	1.18.12	1.18.12	Y	Y	Y
ON	1.18.12	1.18.12	1.18.12	Y	Y	Y
IP	N	N	1.18.12	Y	Y	Y
MASK	N	N	1.18.12	Y	Y	Y
IPB	Y	Y	Y	Y	Y	Y
ITB	Y	Y	Y	Y	Y	Y
IVEL	Y	Y	Y	Y	Y	Y
JLM	Y	Y	Y	Y	Y	Y

Command	ACR1505 (1.18.09)	ACR8020 (1.18.06 Upd09)	ACR9000 (1.19.0)	ACR9030 (1.19.0)	ACR9040 (1.19.0)	Aries Controller
JOG	Y	Y	Y	Y	Y	Y
ABS	Y	Y	Y	Y	Y	Y
ACC	Y	Y	Y	Y	Y	Y
DEC	Y	Y	Y	Y	Y	Y
FWD	Y	Y	Y	Y	Y	Y
HOME	N	N	1.18.15	1.18.15	1.18.15	Y
HOMEVF	N	N	1.18.15	1.18.15	1.18.15	Y
INC	Y	Y	Y	Y	Y	Y
JRK	Y	Y	Y	Y	Y	Y
OFF	Y	Y	Y	Y	Y	Y
REN	Y	Y	Y	Y	Y	Y
RES	Y	Y	Y	Y	Y	Y
REV	Y	Y	Y	Y	Y	Y
SRC	Y	Y	Y	Y	Y	Y
VEL	Y	Y	Y	Y	Y	Y
JRK	Y	Y	Y	Y	Y	Y
KVF	Y	Y	Y	Y	Y	Y
KVI	Y	Y	Y	Y	Y	Y
KVP	Y	Y	Y	Y	Y	Y
LIMIT	Y	Y	Y	Y	Y	Y
FREQ	Y	Y	Y	Y	Y	Y
MULT	Y	Y	Y	Y	Y	Y
SRC WIDTH	Y	Y	Y	Y	Y	Y
LIST	Y	Y	Y	Y	Y	Y
LISTEN	Y	Y	Y	Y	Y	Y
LOCK	Y	Y	Y	Y	Y	Y
LOOK	Y	Y	Y	Y	Y	Y
ANG	Y	Y	Y	Y	Y	Y
MODE	Y	Y	Y	Y	Y	Y
OFF	Y	Y	Y	Y	Y	Y
ON	Y	Y	Y	Y	Y	Y
LOPASS	Y	Y	Y	Y	Y	Y
LRUN	Y	Y	Y	Y	Y	Y
MASK	Y	Y	Y	Y	Y	Y
MASTER	Y	Y	Y	Y	Y	Y
MAXVEL	Y	Y	Y	Y	Y	Y
MBUF	Y	Y	Y	Y	Y	Y
OFF	Y	Y	Y	Y	Y	Y
ON	Y	Y	Y	Y	Y	Y
MEM	Y	Y	Y	Y	Y	Y
MODE	Y	Y	Y	Y	Y	Y
MOV	Y	Y	Y	Y	Y	Y
MSEEK	Y	Y	Y	Y	Y	Y

Command	ACR1505 (1.18.09)	ACR8020 (1.18.06 Upd09)	ACR9000 (1.19.0)	ACR9030 (1.19.0)	ACR9040 (1.19.0)	Aries Controller
MULT	Y	Y	Y	Y	Y	Y
NEW	Y	Y	Y	Y	Y	Y
NORM	Y	Y	Y	Y	Y	Y
NOTCH	Y	Y	Y	Y	Y	Y
NURB	Y	Y	Y	Y	Y	Y
END	Y	Y	Y	Y	Y	Y
MODE	Y	Y	Y	Y	Y	Y
RANK	Y	Y	Y	Y	Y	Y
OFFSET	Y	Y	Y	Y	Y	Y
OOP	Y	N	N	N	N	N
BASE	Y	N	N	N	N	N
OFF	Y	N	N	N	N	N
ON	Y	N	N	N	N	N
OPEN	Y	Y	Y	Y	Y	Y
DTALK	N	N	Y	Y	Y	N
EPLD	N	N	N	Y	Y	N
PASSWORD	Y	Y	Y	Y	Y	Y
OFF	Y	Y	Y	Y	Y	Y
ON	Y	Y	Y	Y	Y	Y
PAUSE	Y	Y	Y	Y	Y	Y
PBOOT	Y	Y	Y	Y	Y	Y
PERIOD	Y	Y	Y	Y	Y	Y
PGAIN	Y	Y	Y	Y	Y	Y
PLC	Y	Y	Y	Y	Y	N
PLS	Y	Y	Y	Y	Y	Y
BASE	Y	Y	Y	Y	Y	Y
DST	Y	Y	Y	Y	Y	Y
FLZ	Y	Y	Y	Y	Y	Y
MASK	Y	Y	Y	Y	Y	Y
OFF	Y	Y	Y	Y	Y	Y
ON RATIO	Y	Y	Y	Y	Y	Y
RES	Y	Y	Y	Y	Y	Y
ROATARY	Y	Y	Y	Y	Y	Y
SRC	Y	Y	Y	Y	Y	Y
PM	—	—	—	—	—	—
ACC	1.18.15	1.18.15	1.18.15	Y	Y	N
DB	1.18.15	1.18.15	1.18.15	Y	Y	N
LIST	1.18.15	1.18.15	1.18.15	Y	Y	N
OFF	1.18.15	1.18.15	1.18.15	Y	Y	N
ON	1.18.15	1.18.15	1.18.15	Y	Y	N
REN	1.18.15	1.18.15	1.18.15	Y	Y	N
SCALE	1.18.15	1.18.15	1.18.15	Y	Y	N
VEL	1.18.15	1.18.15	1.18.15	Y	Y	N

Command	ACR1505 (1.18.09)	ACR8020 (1.18.06 Upd09)	ACR9000 (1.19.0)	ACR9030 (1.19.0)	ACR9040 (1.19.0)	Aries Controller
PPU	Y	Y	Y	Y	Y	Y
PRINT	Y	Y	Y	Y	Y	Y
PROG	Y	Y	Y	Y	Y	Y
PROGRAM	Y	1.18.07	Y	Y	Y	Y
PROM	Y	Y	Y	Y	Y	N
RATCH	Y	Y	Y	Y	Y	Y
MODE	Y	Y	Y	Y	Y	Y
SRC	Y	Y	Y	Y	Y	Y
REBOOT	Y	Y	Y	Y	Y	Y
REM	Y	Y	Y	Y	Y	Y
REN	Y	Y	Y	Y	Y	Y
RES	Y	Y	Y	Y	Y	Y
RESUME	Y	Y	Y	Y	Y	Y
RETURN	Y	Y	Y	Y	Y	Y
ROTARY	Y	Y	Y	Y	Y	Y
ROTATE	Y	Y	Y	Y	Y	Y
ROV	Y	Y	Y	Y	Y	Y
RUN	Y	Y	Y	Y	Y	Y
SAMP	Y	Y	Y	Y	Y	Y
BASE	Y	Y	Y	Y	Y	Y
CLEAR	Y	Y	Y	Y	Y	Y
SRC	Y	Y	Y	Y	Y	Y
TRG	Y	Y	Y	Y	Y	Y
SCALE	Y	Y	Y	Y	Y	Y
SET	Y	Y	Y	Y	Y	Y
SINE	Y	Y	Y	Y	Y	Y
SLDEC	N	N	1.18.15	Y	Y	Y
SLIM	N	N	1.18.15	Y	Y	Y
SLM	N	N	1.18.15	Y	Y	Y
SPLINE	Y	Y	Y	Y	Y	Y
END	Y	Y	Y	Y	Y	Y
MODE	Y	Y	Y	Y	Y	Y
SRC	Y	Y	Y	Y	Y	Y
STEP	Y	Y	Y	Y	Y	Y
STP	Y	Y	Y	Y	Y	Y
STREAM	1.18.11	1.18.11	1.18.11	Y	Y	Y
TIMEOUT	1.18.16	1.18.16	1.18.16	Y	Y	N
SYNC	Y	Y	Y	Y	Y	Y
OFF	Y	Y	Y	Y	Y	Y
ON	Y	Y	Y	Y	Y	Y
MODE	Y	Y	Y	Y	Y	Y

Command	ACR1505 (1.18.09)	ACR8020 (1.18.06 Upd09)	ACR9000 (1.19.0)	ACR9030 (1.19.0)	ACR9040 (1.19.0)	Aries Controller
PROG	Y	Y	Y	Y	Y	Y
SYS	Y	Y	Y	Y	Y	Y
TALK TO	N	N	Y	Y	Y	N
TANG	Y	Y	Y	Y	Y	Y
OFF	Y	Y	Y	Y	Y	Y
ON	Y	Y	Y	Y	Y	Y
TARC	Y	Y	Y	Y	Y	Y
OFF	Y	Y	Y	Y	Y	Y
ON	Y	Y	Y	Y	Y	Y
TLM	Y	Y	Y	Y	Y	Y
TMOV	Y	Y	Y	Y	Y	Y
OFF	Y	Y	Y	Y	Y	Y
ON	Y	Y	Y	Y	Y	Y
VEL	Y	Y	Y	Y	Y	Y
TOV	Y	Y	Y	Y	Y	Y
TRG	Y	Y	Y	Y	Y	Y
TRJ	Y	Y	Y	Y	Y	Y
TROFF	Y	Y	Y	Y	Y	Y
TRON	Y	Y	Y	Y	Y	Y
UNLOCK	Y	Y	Y	Y	Y	Y
VECDEF	Y	Y	Y	Y	Y	Y
VECTOR	Y	Y	Y	Y	Y	Y
VEL	Y	Y	Y	Y	Y	Y
LIMIT	Y	Y	Y	Y	Y	
VER	Y	Y	Y	Y	Y	Y
WHILE/WEND	Y	1.18.07	Y	Y	Y	Y

Command Groups

Axis Limits	
Command	**Description**
ALM	Set stroke limit ‘A’
BLM	Set stroke limit ‘B’
EXC	Set excess error band
HLBIT	Set hardware limit/homing input
HLDEC	Hardware limit deceleration
HLIM	Hardware limit enable
IPB	Set in-position band
ITB	Set in-torque band
JLM	Set jog limits
MAXVEL	Set velocity limits
PM	Position maintenance
SLDEC	Software limit deceleration
SLIM	Software limit enable
SLM	Software positive/negative travel range
TLM	Set torque limits

Character I/O	
Command	**Description**
CLOSE	Close a device
DTALK	Drive Talk
INPUT	Receive data from a device
OPEN	Open a device
PRINT	Send data to a device
TALK TO	Talk to device

Drive Control	
Command	**Description**
DRIVE	Drive report-back
EPLC	Define EPLC

Feedback Control	
Command	**Description**
HSINT	High speed interrupt
INTCAP	Encoder capture
MSEEK	Marker seek operation
MULT	Set encoder multipliers
NORM	Normalize current position
OOP	High speed output
PPU	Set axis pulse/unit ratio
REN	Match position with encoder
RES	Reset or preload encoder
ROTARY	Set rotary axis length

Global Objects	
Command	**Description**
ADC	Analog input control
ADCX	Expansion board analog input
AXIS	Direct axis access
CIP	Ethernet/IP status
DAC	Analog output control
ENC	Quadrature input control
FSTAT	Fast status setup
LIMIT	Frequency limiter
MASTER	Direct master access
PLS	Programmable limit switch
RATCH	Software ratchet
SAMP	Data sampling control

Interpolation	
Command	**Description**
CIRCCW	Counter clockwise circular move
CIRCW	Clockwise circular move
INT	Interruptible move
INVK	Inverse kinematics
MOV	Define a linear move
NURB	NURBs interpolation mode
SINE	Sinusoidal move
SPLINE	Spline interpolation mode
TANG	Tangential move mode
TARC	3-D circular interpolation
TRJ	Start new trajectory

Logic Function	
Command	**Description**
CLR	Clear a bit flag
DWL	Delay for a given period
IHPOS	Inhibit on position
INH	Inhibit on bit high or low
MASK	Safe bit masking
SET	Set a bit flag
TRG	Start move on trigger

Memory Control	
Command	**Description**
CLEAR	Clear memory allocation
DIM	Allocate memory
MEM	Display memory allocation

Nonvolatile	
Command	**Description**
BRESET	Disable battery backup
ELOAD	Load system parameters
ERASE	Clear the EEPROM
ESAVE	Save system parameters
FIRMWARE	Firmware upgrade/backup
FLASH	Create user image in flash
PBOOT	Auto-run program
PROM	Dump burner image

Operating System	
Command	**Description**
ATTACH	Define attachments
CONFIG	Hardware configuration
CPU	Display processor loading
DEF	Display the defined variable
#DEFINE	Define variable
DETACH	Clear attachments
DIAG	Display system diagnostics
ECHO	Character echo control
HELP	Display command list
IP	IP address
MODE	Binary data formatting
PASSWORD	Block uploading programs from board
PERIOD	Set base system timer period
PLC	Switch to a PLC prompt
PROG	Switch to a program prompt
REBOOT	Reboot controller card
STREAM	Display stream name
SYS	Return to system prompt
VER	Display firmware version

Program Control	
Command	**Description**
AUT	Turn off block mode
BLK	Turn on block mode
HALT	Halt an executing program
LIST	List a stored program
LISTEN	Listen to program output
LRUN	Run and listen to a program
NEW	Clear out a stored program
PAUSE	Activate pause mode
REM	Program comment
RESUME	Release pause mode
RUN	Run a stored program
STEP	Step in block mode
TROFF	Turn off trace mode
TRON	Turn on trace mode

Program Flow	
Command	**Description**
BREAK	Exit a program loop
END	End of program execution
ENDP	End program without line numbers
FOR/TO/STEP/NEXT	Relative program path shift
GOSUB	Branch to a subroutine
GOTO	Branch to a new line number
IF / ELSE IF / ELSE / ENDIF	Conditional execution
IF / THEN	Conditional execution
PROGRAM	Beginning of program definition
RETURN	Return from a subroutine
WHILE/WEND	Loop execution conditional

Servo Control	
Command	**Description**
DGAIN	Set derivative gain
DIN	Dead zone integrator negative value
DIP	Dead zone integrator positive value
DWIDTH	Set derivative sample period
DZL	Dead zone inner band
DZU	Dead zone outer band
FBVEL	Set feedback velocity
FFACC	Set feedforward acceleration
FFVC	Feedforward velocity cutoff region
FFVEL	Set feedforward velocity
FLT	Digital filter move
IDELAY	Set integral time-out delay
IGAIN	Set integral gain
ILIMIT	Set integral anti-windup limit
KVF	PV loop feedforward gain
KVI	PV loop integral gain
KVP	PV loop proportional gain
LOPASS	Setup lopass filter
NOTCH	Setup notch filter
PGAIN	Set proportional gain

Setpoint Control	
Command	**Description**
BKL	Set backlash compensation
BSC	Ballscrew compensation
CAM	Electronic cam
GEAR	Electronic gearing
HDW	Hand wheel
JOG	Single axis velocity profile
LOCK	Lock gantry axis
UNLOCK	Unlock gantry axis

Transformation	
Command	**Description**
FLZ	Relative program path shift
OFFSET	Absolute program path shift
ROTATE	Rotate a programmed path
SCALE	Scale a programmed path

Velocity Profile	
Command	**Description**
ACC	Set acceleration ramp
DEC	Set deceleration ramp
F	Set velocity in units/minute
FOV	Set feedrate override
FVEL	Set final velocity
IVEL	Set initial velocity
JRK	Set jerk parameter (S-curve)
LOOK	Look ahead mode
MBUF	Multiple move buffer mode
ROV	Set rapid feedrate override
SRC	Set external time base
STP	Set stop ramp
SYNC	Synchronization mode
TMOV	Set time based move
TOV	Time override
VECDEF	Define automatic vector
VECTOR	Set manual vector
VEL	Set target velocity for a move

Command Syntax

The AcroBASIC programming language accommodate a wide range of needs by providing basic motion control building blocks, as well as sophisticated motion and program flow constructs.

The language comprises simple ASCII mnemonic commands, with each command separated by a command delimiter (carriage return, colon, or line feed). The command delimiter indicates that a command is ready for processing.

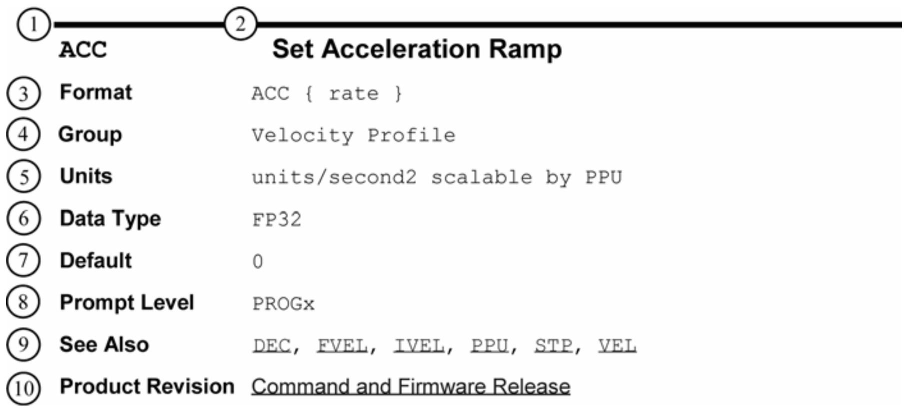
The AcroBASIC programming language uses a parent daughter approach. A parent can have daughter statements; a daughter statement is considered a sub-statement of the parent.

Many parent statements can be issued alone—some provide the current status related to that particular command, others perform an action. For example, issuing the **DIM** command at the system level provides you with a report of the system dimensions. Conversely, issuing the **CLEAR** command at the system level frees the memory allocated to all programs.

You can only issue some parent commands in conjunction with a daughter statement. For example, the **FLASH** command has the **ERASE**, **LOAD**, **IMAGE**, **RES**, and **SAVE** daughter statements. Therefore, you can issue the **FLASH ERASE**, **FLASH LOAD**, **FLASH IMAGE**, **FLASH RES**, and **FLASH SAVE** commands but not **FLASH** by itself.

Description of Format

Each parent or daughter command shows the necessary elements to correctly use that command. The following describes how to interpret the command format presented to you in this guide:



1	**ACC**	2 **Set Acceleration Ramp**
3	**Format**	`ACC { rate }`
4	**Group**	`Velocity Profile`
5	**Units**	`units/second2 scalable by PPU`
6	**Data Type**	`FP32`
7	**Default**	`0`
8	**Prompt Level**	`PROGx`
9	**See Also**	`DEC, FVEL, IVEL, PPU, STP, VEL`
10	**Product Revision**	Command and Firmware Release

1. **Mnemonic Code:** The ASCII command.
2. **Name:** A short description of the command.
3. **Format:** Indicates the proper syntax and arguments for the command. For descriptions of the arguments, see argument descriptions for each command.
4. **Group:** The functional group to which the command belongs.
5. **Units:** Indicates the units of measurement required by the argument(s) in the command syntax.

6. **Data Type:** Indicates the class of data required by the argument(s).
7. **Default:** Indicates the setting or value automatically selected unless you specify a substitute.
8. **Prompt Level:** Indicates the communication level at which you can use the command. For more information, see Communication Levels.
9. **See Also:** Indicates commands related or similar to the command you are reviewing.
10. **Product Revision:** To determine whether the command applies to your specific ACR series controller and firmware revision, see the Command and Firmware Release table.

Arguments and Syntax

The syntax of an AcroBASIC command shows all the components necessary to use the command. Commands can contain required and optional arguments. They also contain a number of symbols:

- Braces { }—arguments that are optional. Do not type the braces in your code.
- Parentheses ( )—arguments that are optional, and must appear within the parentheses in your code. Also used to indicate variables and expressions. If replacing a constant with a variable or parametric equation, use parentheses to "contain" the variable/equation. Signed (-) or (+) constants must be in parentheses.
- Commas (,)—delimiters between arguments in specific commands. In addition, select commands use commas to control spacing and line feeds. To understand the separator's specific use in a command, refer to the command's format and description.
- Semicolons (;)—delimiters between arguments in specific commands. In addition, select commands use semicolons to control spacing and line feeds. To understand the separator's specific use in a command, refer to the command's format and description.
- Vertical bar (|)—indicates a choice between arguments.
- Slash mark (/)—signifies an incremental move in select commands.
- Quotes (" ")—arguments within the quotes must appear within quotes in your code.
- Number sign (#)—device arguments following number signs must include the number sign in your code.
- Ellipsis (…)—arguments can be given for multiple axes

The following examples illustrate how to interpret common syntax:

Example 1

ACC {*rate*}

In the **ACC** command, the lower case word *rate* is an argument. Arguments act as placeholders for data you provide. If an argument appears in braces or parentheses, the argument is optional.

For example, the following sets the acceleration ramp to 10,000 units per second^2.

```
ACC 10000
```

When you issue a command without an optional argument, the controller reports back the current setting. Not all commands report back, and some require you to specify an axis. For example, the following reports the current acceleration rate in program 0.

```
P00>ACC

10000
```

Example 2

FBVEL {*axis* **{***value***}} {***axis* **{***value***}} ...**

Optional arguments can nest. This provides the flexibility to set data for or receive reports on multiple axes. For example, the following sets the velocity feedback gain for axes X and Y to 0.0001 and 0.0002 respectively.

```
FBVEL X 0.0001 Y 0.0002
```

Because the **FBVEL** command can report on multiple axes, you specify at least one axis on which the controller is to report back.

```
P00>FBVEL X

0.0001

P00>FBVEL X Y

0.0001

0.0002
```

Example 3

IPB {*axis* **{***value***}} {***axis* **{(***value1***,** *value2***)}} ...**

The AcroBASIC language provides programming shortcuts. You can set positive and negative values for commands using one argument. If the values differ, you can use two arguments. The command format illustrates when this is possible. For example, the following sets the in-position band for axis X to ±0.05 and for axis y to 3 and –1.

```
IPB X 0.05 Y(3, -1)
```

Notice that the two values for axis Y are given inside parentheses and separated by a comma, as shown in the format of the command.

Example 4

HALT {*PROGx* **|** *PLCx* **|** *ALL***}**

The vertical bar indicates a choice between arguments. For example, the **HALT** command lets you stop a user program, a PLC program, or all programs.

```
HALT PROG0

HALT PLC5

HALT ALL
```

Commands

The following pages contain details and usage information for each command.

Example Code Conventions

Examples that include code are provided throughout most of the ACR Series documentation to illustrate a concept, supply model code samples, or to show multiple ways to employ the commands.

The example code may include the terminal prompt or configuration code if it is necessary for clarity. Example code is complete only as far as conveying information about the discussion, and configuration and other information may need to be added in order for the code to be of use in an actual application.

In ACR Series example code, Axis0 is the X axis, and Axis1 is the Y axis, unless otherwise specified.

Numeric examples of parameters and bits (flags) are called out in parentheses, and are typically for Axis0, Master0, or Program0.

Example

"Setting gearing acceleration to 0.0 (default) or setting the Gear Lock flag (e.g., Bit Index 13 of the Primary Axis Flags) will cause an immediate lock."

ACC Set Acceleration Ramp

Format	ACC {*rate*}
Group	Velocity Profile
Units	units/second^2 scalable by PPU
Data Type	FP32
Default	20000
Prompt Level	PROGx
Report Back	Yes **To view, type ACC**
See Also	DEC, FVEL, IVEL, PPU, STP, VEL
Related Topics	Master Parameters (0-7) (8-15)
Product Revision	Command and Firmware Release

Description

Sets the master acceleration for ramping from lower to higher speeds for coordinated motion.

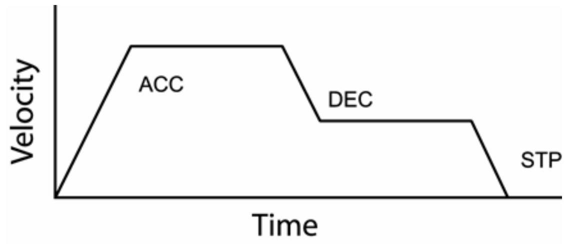
Arguments

rate

Optional. Sets the acceleration rate for the master profile.

Remarks

The following illustrates **ACC**/**DEC**/**STP** use:



Disabling Acceleration Ramp

With the *rate* set to zero, you can completely disable the acceleration ramp. There are specific circumstances where you want to set the acceleration rate to zero, such as servo tuning or sinusoidal motion (**SINE** command).

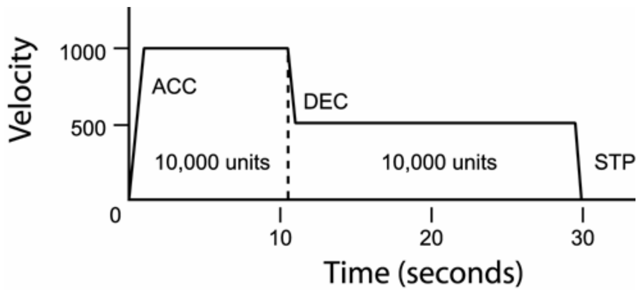
If the motor needs to speed up (for example, sending **FOV**, thereby changing the motion profile velocity), the motion profiler uses infinite acceleration, which is nearly instantaneous.

- ► To disable the acceleration ramp, set the *rate* to zero.

Warning — We do not recommend setting the acceleration rate to zero. Instantaneous acceleration can create excessive load on the mechanics of an application. You must well understand the needs of your application.

Example

The following example sets up a simple motion profile. The first move accelerates to velocity and moves 10,000 incremental units. At the start of the second move, the motor ramps down to the new velocity using the previously stated deceleration rate. At the end of the second move, the stop ramp ends motion.



```
ACC 1000 DEC 1000 STP 0 VEL 1000
X/10000   : REM move 10,000 incremental units
STP1000 VEL500
X/10000   : REM move 10,000 incremental units
```

ADC Analog Input

Format	ADC {*index*} *command* {*data*}
Group	Global Objects
Units	None
Data Type	N/a
Default	N/a
Prompt Level	SYS, PROGx, PLCx
See Also	ADCX, AXIS, DAC, ENC, PERIOD
Related Topics	Object Parameters
Product Revision	Command and Firmware Release

Description

This command is used alone or in conjunction with a second command to control the optional analog input module.

Remarks

By default, the analog input module converts eight single-ended ±10 volt signals, using default positive inputs and treating the analog ground pin as the negative input for all channels. Optionally, the channels can be read as differential pairs by redirecting positive and negative input signals from any of the eight analog input pins.

Issuing an **ADC** command without an argument will display the current general setting for the **ADC**. Issuing an **ADC** command to an **ADC** channel without an argument will display the current setting for that **ADC** channel.

All enabled analog inputs are updated at the servo period (**PERIOD**). For more information about connection information and the analog inputs, see your hardware installation guide.

The following is a list of valid command combinations:

ADC FLT: Set lopass filter.

ADC GAIN: Set analog input gain.

ADC MAX: Set the number of ADCs (16-bit ADC only).

ADC MODE: Select the firmware mode.

ADC NEG: Select negative channel.

ADC OFF: Disable **ADC** update.

ADC OFFSET: Set analog input offset.

ADC ON: Enable **ADC** update.

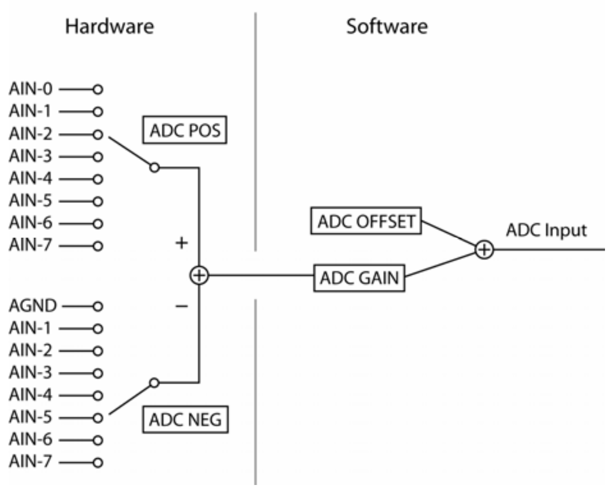
ADC POS: Select positive channel.

ADC SCALE: Set the physical gain of PGA (16-bit ADC only).

ADC SRC: Set the parameter as the alternative to ANI source.

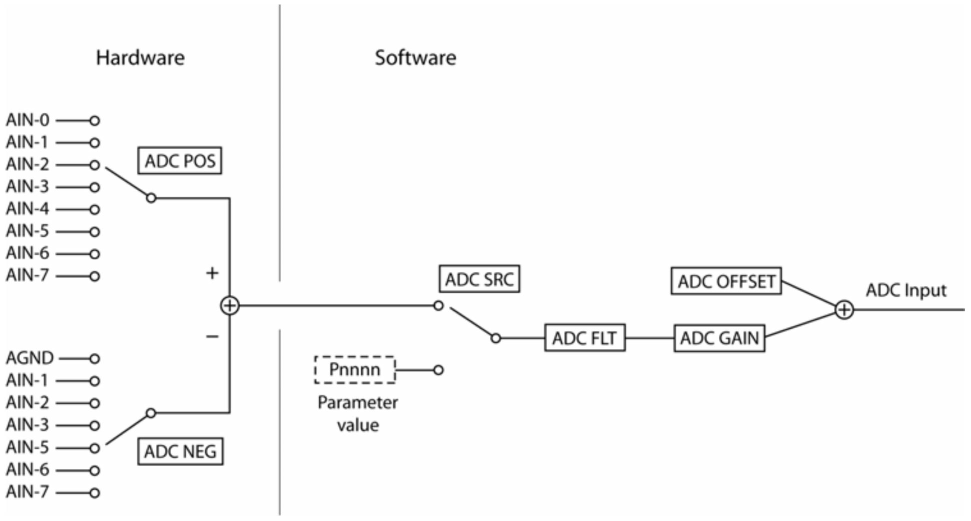
Block Diagram—ACR1505 and ACR8020

The following block diagram illustrates a single analog input channel:



Block Diagram—ACR9000 only

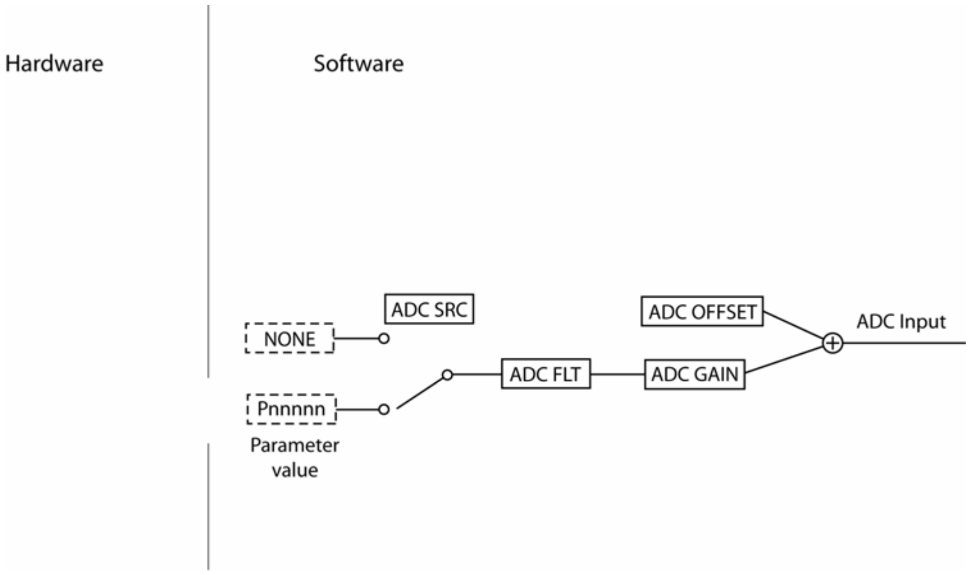
The following block diagram illustrates a single analog input channel for the ACR9000:



Parameter value in the above drawing is any system parameter (P4096 and up). The switch at ADC SRC is in the NONE position (using onboard ADCs).

Block Diagram—ACR9030 and ACR9040

The following block diagram illustrates a single ADC input channel for the ACR9030/9040:



There is no onboard ADC input in the ACR9030 or ACR9040. See Example 3 below.

Examples

Example 1

```
P15>ADC
ADC OFF
ADC MODE 1
ADC MAX 8
```

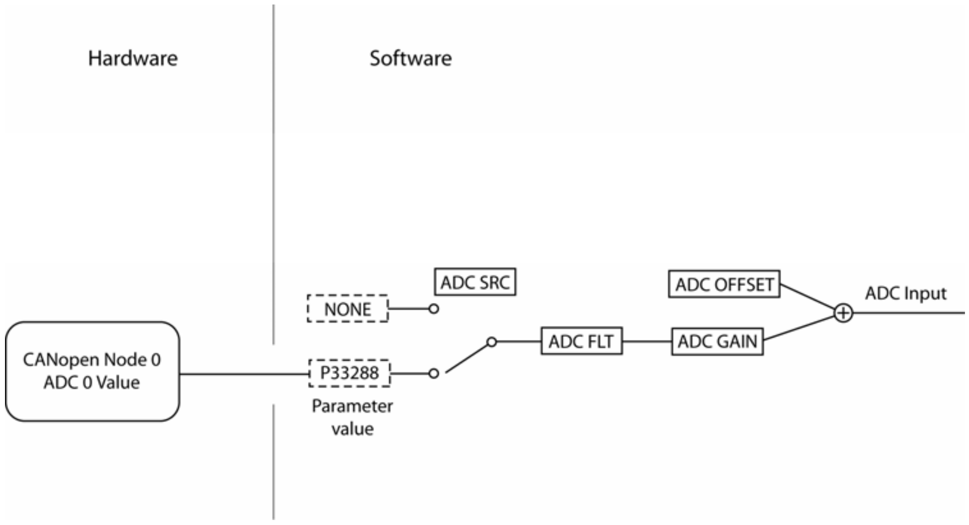
Example 2

```
P15>ADC0
ADC OFF
ADC MODE 1
ADC MAX 8
ADC0 SCALE 10
ADC0 GAIN 10
ADC0 OFFSET 0
ADC0 POS 0
ADC0 NEG 0
ADC0 FLT 0
ADC0 SRC None
```

Example 3

This example configures a single ADC input, ADC0, mapped to a CANopen analog input.

```
ADC MAX 1
ADC0 GAIN 10
ADC0 OFFSET 0
ADC0 FLT 0
ADC0 SRC P33288
ADC ON
```



ADC FLT Set Lopass Filter

		Product	Revision
Format	ADC *number* FLT {*cutoff frequency*}		
Group	Global Objects	ACR1505	n/a
Units	Hz	ACR8020	n/a
Data Type	Float	ACR9000	1.18.16
Default	0= Filter Off	ACR9030	n/a
Prompt Level	SYS, PROGx	ACR9040	n/a
See Also	ADC, ADCX, AXIS, DAC, ENC	Aries CE	n/a
Related Topics	N/A		

Description

This command sets the cutoff frequency of 2nd order low-pass Butterworth digital filter. The ADC signal will pass through this low pass filter. This filter will reject high frequency noise.

Remarks

By default this filter is off. Setting the cutoff of frequency to zero will turn of the filter. Issuing just the **ADC** *n* **FLT** command will display the current setting of the filter.

NOTE: The maximum frequency is dependant on the servo period.
$0 < fc < 0.5*(1/T)$
Where: fc = Cutoff Frequency; T = Servo Period (Default is 500 microseconds)

Example

The example sets the cutoff frequency of the filter to 500 Hz.

```
ADC 0 FLT 500
ADC 0 FLT
500
```

ADC GAIN Set Analog Input Gain

Format	ADC *index* GAIN {*gain*}
Group	Global Objects
Units	volts / input unit
Data Type	FP32
Default	10
Prompt Level	SYS, PROGx, PLCx
See Also	ADC, ADCX, AXIS, DAC, ENC
Related Topics	N/A
Product Revision	Command and Firmware Release

This command sets the software gain for analog conversion. Issuing an **ADC GAIN** command with no argument will display the current setting. The default **ADC** gain value is 10.0 volts / input unit.

When **ADC** updating is enabled, the readings from the analog input module are internally scaled to generate a base number of ±1.0 input units. This number is multiplied by the **ADC GAIN** setting and then the **ADC OFFSET** value is added. The result is stored in the ADC input parameter.

Example

The first example sets the gain on **ADC** 2 to 9.985 volts = full input unit. The second example will show 4095 = full scale input.

```
ADC 2 GAIN 9.985
ADC 2 GAIN 4095
```

ADC MAX Set the Number of ADC Inputs (16-Bit ADC Only)

Format	ADC MAX {*number*}
Group	Global Objects
Units	None
Data Type	N/A
Default	8
Prompt Level	SYS, PROGx, PLCx
See Also	ADC, AXIS, DAC, ENC
Related Topics	N/A
Product Revision	Command and Firmware Release

NOTE: This command applies only to the 16-bit ADC.

This command sets the number of ADC inputs that will be sampled during the servo-interrupt period. Issuing an **ADC MAX** command with no argument will display the current number of ADC inputs selected. The default number is 8. This command is only for 16-bit ADC operation.

NOTE: The 12-bit ADC firmware always samples all 8 ADC inputs. This command is not supported by the ACR2000

Example

The following example sets the number of ADC inputs to 5:

```
ADC MAX 5
```

ADC MODE Select the 12-Bit/16-Bit Firmware Mode

		Product	Revision
Format	ADC MODE {*mode*}	ACR1505	all
Group	Global Objects	ACR8020	all
Units	None	ACR9000	all
Data Type	N/A	ACR9030	n/a
Default	0	ACR9040	n/a
Prompt Level	SYS, PROGx, PLCx	Aries CE	n/a
See Also	ADC, ADCX, AXIS, DAC, ENC		
Related Topics	Miscellaneous Parameters		

Description

This command sets the firmware mode for the 12-bit or 16-bit ADC module on the ACR1200, ACR1500, ACR1505, ACR8000, ACR8010, ACR8020 boards. Issuing an **ADC MODE** command with no argument will display the current mode. The default mode is for 12-bit ADC. The mode is set based on the type of ADC module installed, and is not interchangeable between modules. The **ADC MODE** can be saved using the **ESAVE** command.

Remarks

The following table shows the modes:

Mode	Description
0	STD 12-bit ADC
1	16-bit ADC

Example

The following example sets mode to 16-bit ADC:

```
ADC MODE 1
```

ADC NEG Select Negative Channel

Format	ADC *index* NEG {*channel*}
Group	Global Objects
Units	None
Data Type	N/A
Default	0
Prompt Level	SYS, PROGx, PLCx
See Also	ADC, ADCX, AXIS, DAC, ENC
Related Topics	Miscellaneous Parameters
Product Revision	Command and Firmware Release

This command sets the negative input for differential analog conversion. Issuing an **ADC NEG** command with no argument will display the current setting. The default negative channel is 0 for each conversion.

Always refer to the appropriate hardware manual for pin-out information.

The following table shows the relationship between the "channel" and negative input. Note that channel 0 attaches to analog ground instead of input 0 like the positive channel assignment does.

ACR1200, ACR1500, ACR1505, ACR2000, ACR8000, ACR8010, ACR8020	
Channel	**Pin Name**
0	AGND
1	AIN1
2	AIN2
3	AIN3
4	AIN4
5	AIN5
6	AIN6
7	AIN7

ACR9000 only	
Channel	**Pin Name**
0	AGND
1	AIN1
2	AIN2
3	AIN3
4	AGND
5	AIN5
6	AIN6
7	AIN7

The following table shows the relationship between the ADCX "channel" and negative input (only for the ACR1505 and ACR8020):

Channel	Pin Name
0	AGND
1	AIN9
2	AIN10
3	AIN11
4	AIN12
5	AIN13
6	AIN14
7	AIN15

Example

The following example sets the negative input of **ADC** 4 to channel 3:

```
ADC 4 NEG 3
```

Setup example for four differential analog inputs:

ADC Setting	ADC Input Number	ADC Value
ADC 0 POS 0 ADC 0 NEG 1	Differential input #1	P6408
ADC 1 POS 2 ADC 1 NEG 3	Differential input #2	P6424
ADC 2 POS 4 ADC 2 NEG 5	Differential input #3	P6440
ADC 3 POS 6 ADC 3 NEG 7	Differential input #4	P6456

ADC OFF Disable ADC Update

Format	ADC OFF
Group	Global Objects
Units	None
Data Type	N/A
Default	
Prompt Level	SYS, PROGx, PLCx
See Also	ADC, ADCX, AXIS, DAC, ENC
Related Topics	N/A
Product Revision	Command and Firmware Release

This command disables the update of the analog input module. Note that this command controls the update of all analog inputs and does not have an *index* argument as other **ADC** commands.

Example

The following example disables **ADC** updating:

```
ADC OFF
```

ADC OFFSET Set Analog Input Offset

Format	ADC *index* OFFSET {*offset*}
Group	Global Objects
Units	volts
Data Type	FP32
Default	0.0
Prompt Level	SYS, PROGx, PLCx
See Also	ADC, ADCX, AXIS, DAC, ENC
Related Topics	N/A
Product Revision	Command and Firmware Release

This command sets the software offset for analog conversion. Issuing an **ADC OFFSET** command with no argument will display the current setting. The default **ADC** offset value is 0.0 volts.

When **ADC** updating is enabled, the readings from the analog input module are internally scaled to generate a base number of ±1.0 input units. This number is multiplied by the **ADC GAIN** setting and then the **ADC OFFSET** value is added. The result is stored in the **ADC** input parameter.

Example

The following example sets the offset on **ADC** 2 to 0.012 volts:

```
ADC 2 OFFSET 0.012
```

ADC ON Enable ADC Update

Format	ADC ON
Group	Global Objects
Units	None
Data Type	N/A
Default	
Prompt Level	SYS, PROGx, PLCx
See Also	ADC, ADCX, AXIS, DAC, ENC
Related Topics	N/A
Product Revision	Command and Firmware Release

This command enables the update of the analog input module. Note that this command controls the update of all analog inputs and does not have an *index* argument as other **ADC** commands.

Example

The following example enables **ADC** updating:

```
ADC ON
```

ADC POS Select Positive Channel

Format	ADC *index* POS {*channel*}
Group	Global Objects
Units	None
Data Type	N/A
Default	See below
Prompt Level	SYS, PROGx, PLCx
See Also	ADC, ADCX, AXIS, DAC, ENC
Related Topics	Miscellaneous Parameters
Product Revision	Command and Firmware Release

This command sets the positive input for differential analog conversion. Issuing an **ADC POS** command with no argument will display the current setting. The default positive channel is equal to the ADC index number.

Always refer to the appropriate hardware manual for pin-out information.

The following table shows the relationship between the ADC "channel" and positive input:

Channel	Pin Name
0	AIN0
1	AIN1
2	AIN2
3	AIN3
4	AIN4
5	AIN5
6	AIN6
7	AIN7

The following table shows the relationship between the ADCX "channel" and positive input (only for the ACR1505 and ACR8020):

Channel	Pin Name
0	AIN8
1	AIN9
2	AIN10
3	AIN11
4	AIN12
5	AIN13
6	AIN14
7	AIN15

NOTE: An ADC channel cannot reference an input from the expansion axis board and vice versa.

Example

The following example sets the positive input of ADC 4 to channel 2:

```
ADC 4 POS 2
```

ADC SCALE Set the Physical Gain of PGA (16-Bit ADC Only)

Format	ADC *index* SCALE {*scale*}
Group	Global Objects
Units	None
Data Type	N/A
Default	1
Prompt Level	SYS, PROGx, PLCx
See Also	ADC, ADCX, AXIS, DAC, ENC
Related Topics	N/A
Product Revision	Command and Firmware Release

NOTE: This command applies only to the 16-bit ADC.

Unlike the 12-bit ADC module, the 16-bit ADC module has a built-in "Programmable Gain Amplifier" (PGA). This allows the user to scale the input signal to match four (4) ranges of input levels. This range can be selected individually for each channel. This way, the entire range of 16 bits can be applied to read a ±1.25V, ±2.5V, ±5V, and ±10V signal.

This command sets the physical gain of the PGA. Issuing an **ADC SCALE** command with no argument will display the current scale. The default gain is 1. This command is only for 16-bit ADC.

The following table shows the relationship between the gain and scale:

Scale	Gain
10	1
5	2
2.5	4
1.25	8

Example

The following example sets the physical gain of ADC 1 to 2:

```
ADC 1 SCALE 5
```

ADC SRC Set Source Parameter

Format	ADC *number* SRC {*input_source*}	**Product**	**Revision**
Group	Global Objects	ACR1505	n/a
Units	N/A	ACR8020	n/a
Data Type	N/A	ACR9000	1.24
Default	None	ACR9030	1.24
Prompt Level	SYS, PROGx	ACR9040	1.24
See Also	ADC, ADCX, AXIS, DAC, ENC	Aries CE	n/a
Related Topics	N/A		

Description

Sets the designated parameter as the source of input for the ADC value.

Remarks

If a parameter is selected, the value of that parameter is used as the input to the **ADC** filter, the **ADC GAIN**, and **ADC OFFSET** instead of the value of the ANI hardware. Please refer to the diagram in the ADC command page. Any LONG or FP32 parameter may be used, although it may be most useful to select a parameter that represents a physical input, such as a CAN analog input.

By default the source is "none." Setting the source to none will return the source to the ANI hardware. Issuing just the **ADC SRC** command will display the current setting of the source.

Examples

EXAMPLE 1

The following sets the input of ADC0 to the value of P34304.

```
SYS>ADC 0 SRC P34304
SYS>ADC 0 SRC
P34304
```

EXAMPLE 2

This sets the input of ADC0 back to none.

```
SYS> ADC0 SRC NONE
SYS> ADC0 SRC
NONE
```

ADCX Expansion Board Analog Input

Format	ADCX
Group	Global Objects
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	ADC, AXIS, DAC, ENC
Related Topics	N/A
Product Revision	Command and Firmware Release

This command displays the current setting of the ADC module on an 8-axis expansion board (available on the ACR1505 and ACR8020).

Example

```
ADCX
```

ADCX MODE 12 or 16-Bit Firmware Mode on Expansion Board

Format	ADCX MODE {*mode*}
Group	Global Objects
Units	N/A
Data Type	N/A
Default	0
Prompt Level	SYS, PROGx, PLCx
See Also	ADC, AXIS, DAC, ENC
Related Topics	N/A
Product Revision	Command and Firmware Release

This command sets the firmware mode for the 12-bit or 16-bit ADC module on the 8-axis expansion board (available on the ACR1505 and ACR8020). Issuing an **ADCX MODE** command with no argument will display the current mode. The default mode is for 12-bit ADC. The mode is set based on the type of ADC module installed, and is no interchangeable between modules. The **ADCX MODE** can be saved using the **ESAVE** command.

The following table shows the modes:

Mode	Description
0	STD 12-bit ADC
1	16-Bit ADC

Example

The following example sets mode to 16-bit ADC on the expansion board

```
ADCX MODE 1
```

ADCX MAX Number of ADC Inputs on Expansion Board (16-Bit Only)

Format	ADCX MAX {*number*}
Group	Global Objects
Units	N/A
Data Type	N/A
Default	8
Prompt Level	SYS, PROGx, PLCx
See Also	ADC, AXIS, DAC, ENC, PERIOD
Related Topics	N/A
Product Revision	Command and Firmware Release

NOTE: This command applies only to the 16-bit ADC.

This command sets the number of ADC inputs that will be sampled during the servo interrupt period on the 8 axis expansion board (available on the ACR1505 and ACR8020). Issuing an **ADCX MAX** command with no argument displays the current number of ADC inputs selected. The default number is 8. This command is only for 16-bit operation. The **ADC MAX** can be saved using the **ESAVE** command.

Example

The following example sets the number of ADC inputs to 5 on the expansion board:

```
ADCX MAX 5
```

ALM Set Stroke Limit 'A'

Format	ALM {*axis* {*value*}} {*axis* {(*high*, *low*)}} ...
Group	Axis Limits
Units	Units scalable by PPU
Data Type	FP32
Default	0.0
Prompt Level	PROGx
See Also	AXIS, BLM, PPU
Related Topics	Axis Flags (0-7)(8-15), Secondary Axis Flags (0-7) (8-15), Axis Parameters (0-7) (8-15), Master Flags (0-7) (8-15)
Product Revision	Command and Firmware Release

This command sets the command position (current position) limits monitored by the "A limit" flags. When the command position of a given axis is within these limits, the appropriate flag is set. Otherwise, the flag is cleared. For masters, the flag is set if all of its slaves are within their limits. This is the only reaction the controller gives to exceeding the limit. It is up to the programmer to write code that will react to the flags in an appropriate manner.

Issuing the **ALM** command to an axis without an argument displays the current positive and negative limits for that axis. Issuing the command with a single argument sets the positive limit to "value" and the negative limit to minus "value". Issuing the command with two arguments sets the positive limit to "high" and the negative limit to "low". The default for both is 0.0 for all axes.

Example

The following example sets different positive and negative "A limits" for X, Y and Z axes.

```
ALM X(10,-10) Y(30,-20) Z(5,0)
```

ATTACH Define Attachments

Format	ATTACH {*command*}
Group	Operating System
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	See below
See Also	AXIS, DETACH, MASTER, PROG
Related Topics	N/A
Product Revision	Command and Firmware Release

This command is used along with a second command to define how programs, masters, axes, signals, and feedbacks are attached to one another. Issuing an **ATTACH** without the optional *command* argument displays the attachments to the current program or, if issued from the system level, all attachments to all programs.

The following is a list of valid **ATTACH** command combinations:

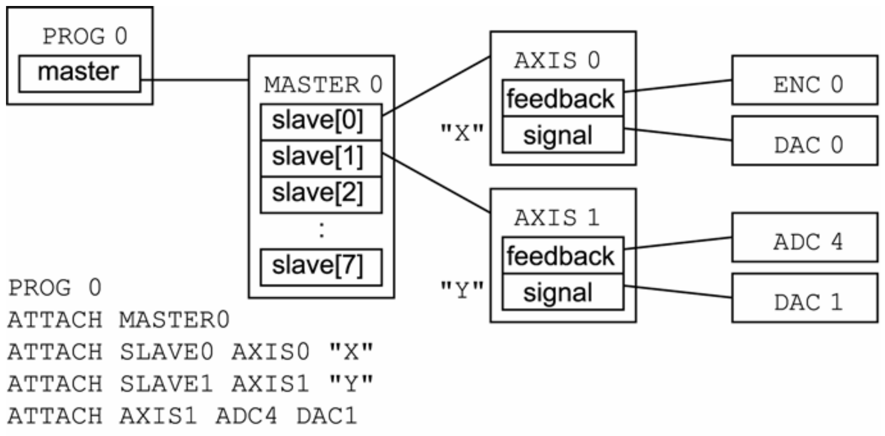
ATTACH AXIS: Attach signal and feedback to axis.

ATTACH MASTER: Attach master to program.

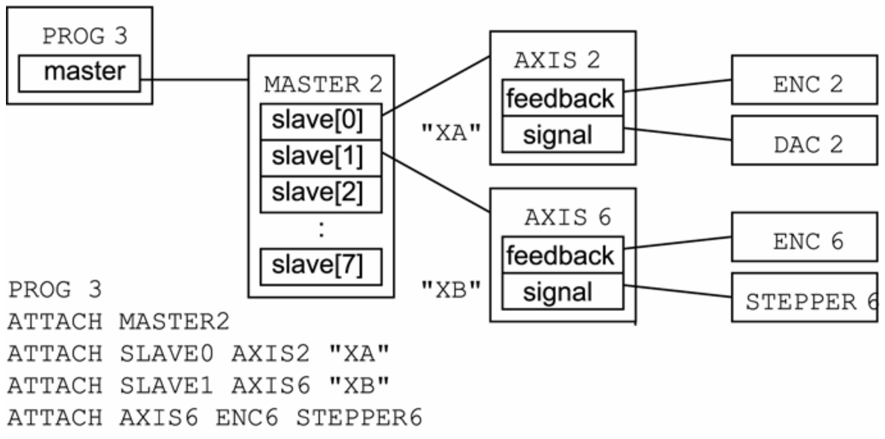
ATTACH SLAVE: Attach axis to master.

Block Diagram

The following block diagram illustrates some sample attachments:



```
PROG 0
ATTACH MASTER0
ATTACH SLAVE0 AXIS0 "X"
ATTACH SLAVE1 AXIS1 "Y"
ATTACH AXIS1 ADC4 DAC1
```



```
PROG 3
ATTACH MASTER2
ATTACH SLAVE0 AXIS2 "XA"
ATTACH SLAVE1 AXIS6 "XB"
ATTACH AXIS6 ENC6 STEPPER6
```

ATTACH AXIS Attach Axis to Signal and Feedback

Format	ATTACH AXIS {*axis* {*position* {*signal* {*velocity*}}}}
Group	Operating System
Units	None
Data Type	N/A
Default	See below
Prompt Level	SYS, PROGx, PLCx
See Also	ATTACH, CONFIG, EPLC OFF, EPLC ON, FBVEL,
Related Topics	N/A
Product Revision	Command and Firmware Release

This command defines the attachment of position feedback and command signal output for a given axis. If the **ATTACH AXIS** command is issued without the optional arguments, the current attachments for all axes are displayed.

The default *position* attachment is *ENC encoder*, where *encoder* is equal to the index of the axis. The following are valid position feedback attachments:

- *EPLD epld:* ETHERNET Powerlink feedback
- *ENC encoder:* Quadrature encoder feedback.
- *ADC adc:* Analog position feedback.
- *STEPPER stepper:* Open loop stepper feedback.

NOTE: When using the *epld* argument, do not include additional arguments in the **ATTACH AXIS** statement.

The default *signal* attachment is *DAC dac*, where *dac* is equal to the index of the axis. The following are valid signal output attachments:

- *DAC dac:* Analog voltage output.
- *STEPPER stepper:* Step and direction output.

The default *velocity* attachment is *ENC encoder*, where *encoder* is equal to the index of the axis. The velocity of this item (derivative) is multiplied by the **FBVEL** setting and subtracted from the control signal. The following are valid velocity feedback attachments:

- *ENC encoder:* Quadrature velocity feedback.
- *ADC adc:* Analog velocity feedback.

Example 1

This attaches ENC 5 as position feedback, DAC 6 as the command signal output, and ADC 7 as velocity feedback on AXIS 4:

```
ATTACH AXIS4 ENC5 DAC6 ADC7
```

Example 2

This attaches no position feedback, STEPPER3 as the command signal output and no velocity feedback on AXIS7:

```
ATTACH AXIS7 STEPPER3 STEPPER3
```

NOTE: When attaching a stepper axis, do not specify a velocity feedback.

Example 3

This attaches the following to a two-axis ACR9030:

ENC0 as position feedback and DAC0 as the command signal output on AXIS 0;

ENC1 as position feedback and DAC1 as the command signal output on AXIS 1;

Drive EPLD0 on Axis2;

Drive EPLD1 on Axis3; and

Drive EPLD2 on Axis 4.

When *epld* (ETHERNET Powerlink drive) is the first argument after **ATTACH AXIS**, no additional arguments in the **ATTACH AXIS** statement are accepted.

```
ATTACH AXIS0 ENC0 DAC0
ATTACH AXIS1 ENC1 DAC1
ATTACH AXIS2 EPLD0
ATTACH AXIS3 EPLD1
ATTACH AXIS4 EPLD2
```

ATTACH MASTER Attach Master to Program

Format	ATTACH MASTER *master*
Group	Operating System
Units	None
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	ATTACH, DETACH, PROG
Related Topics	N/A
Product Revision	Command and Firmware Release

This command attaches a master to the current program. Each master has 16 internal slots that serve as attachment points for axes. This command must be issued from a program prompt. An error will be generated if the master is attached to another program.

You can only attach one master to a program. For more information about masters, see System Attachments on page 24.

Example

The following example attaches master 2 to program 0:

```
PROG0
ATTACH MASTER2
```

ATTACH SLAVE Attach Slave to Axis

Format	ATTACH SLAVE *slave* AXIS *axis* "*name*"
Group	Operating System
Units	None
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	ATTACH, DETACH
Related Topics	N/A
Product Revision	Command and Firmware Release

This command attaches an axis to the current master. The "slave" is an internal slot in the master that the axis is attached to. The "name" is a one to four character alpha string. An error will be generated if another axis is already attached to the slave or if the given axis is attached elsewhere.

For more information about masters, see System Attachments on page 24.

Example

The following example attaches axes 3 and 4 to the current master as "X" and "Y":

```
ATTACH SLAVE0 AXIS3 "X"
ATTACH SLAVE1 AXIS4 "Y"
```

AUT Turn Off Block Mode

Format	AUT {PROG *number* | ALL}
Group	Program Control
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	BLK, PROG, STEP
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Turns off block mode for the selected program.

Arguments

PROG *number* | ALL

Remarks

Block mode is not applicable to PLC programs.

When you use **AUT** (auto), the program's "block control" bit is cleared. To continue normal operation after the **AUT** command, issue a **STEP** command or set the "step request" bit. The controller then detects that "block control" is no longer active and clears the "block mode" bit.

The **AUT PROG** command will turn off block mode (**BLK** command) for the indicated program, and the **AUT ALL** command will turn off block mode for all programs. These commands can be issued from anywhere in the system, including programs.

Example

From the Terminal

The following turns off block mode in program 0:

```
P00> AUT
STEP
```

The following turns off block mode in program 0 from the system prompt:

```
SYS> PROG0
P00> AUT
P00> STEP
```

The following turns off block mode in program 0 while remaining in program 1:

```
SYS> PROG1
P01> AUT PROG0
P01> STEP PROG0
```

In a Program

The following is a subroutine that activates the **BLK** command to halt automation, then steps through the next three lines of code, then reactivates the automation:

```
_Blocksub
BLK
STEP
STEP
STEP
AUT
END
```

Example Bits

Flag Parameter 4128 For Program 0	
Block Control	1040
Block Mode	1042
Step Request	1045

AXIS Direct Axis Access

Format	AXIS *index command* {*data*}
Group	Global Objects
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	DAC, ENC, MASTER
Related Topics	Axis Flags (0-7)(8-15),
Product Revision	Command and Firmware Release

This command allows direct access to an axis without having to use its name, or to another program to which the axis is not attached. The axis does not have to be attached to a master. In general, the *command* argument is any command or command pair that would normally be followed by an axis name and data. These commands include those from the axis limits, feedback control, servo control, and setpoint control groups.

Two other direct axis commands unique to this access mode are:

AXIS OFF: Disable servo loop.

AXIS ON: Enable servo loop.

These commands turn on and off the servo loop associated with an axis without having to use the bit flags designated for that purpose. Turning off unnecessary servo loops will reduce **CPU** load and improve system performance.

NOTE: You can undo these commands at any prompt level.

Example

The following example sets the proportional gain on AXIS 3 to 0.001, jogs AXIS 2 to an absolute jog position of 2.5 units, and disables the AXIS 7 servo loop:

```
AXIS3 PGAIN 0.001
AXIS2 JOG ABS 2.5
AXIS7 OFF
```

AXIS OFF Disable Axis

Format	AXIS *index command* {*data*}
Group	Global Objects
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	DAC, ENC, MASTER
Related Topics	N/A
Product Revision	Command and Firmware Release

This command disables the servo loop associated with an axis without using the bit flag designated for this purpose. Turning off unnecessary servo loops reduce CPU load and improve system performance.

Example

The following example enables the AXIS 5 servo loop, sets the proportional gain on AXIS 3 to 0.001, jogs AXIS 2 to an absolute jog position of 2.5 units, and disables the AXIS 7 servo loop:

```
AXIS5 ON
AXIS3 PGAIN 0.001
AXIS2 JOG ABS 2.5
AXIS7 OFF
```

AXIS ON Enable Axis

Format	AXIS *index command* {*data*}
Group	Global Objects
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	DAC, ENC, MASTER
Related Topics	N/A
Product Revision	Command and Firmware Release

This command enables the servo loop associated with an axis without using the bit flag designated for this purpose.

Example

The following example enables the AXIS 5 servo loop, sets the proportional gain on AXIS 3 to 0.001, jogs AXIS 2 to an absolute jog position of 2.5 units, and disables the AXIS 7 servo loop:

```
AXIS5 ON
AXIS3 PGAIN 0.001
AXIS2 JOG ABS 2.5
AXIS7 OFF
```

BKL Set Backlash Compensation

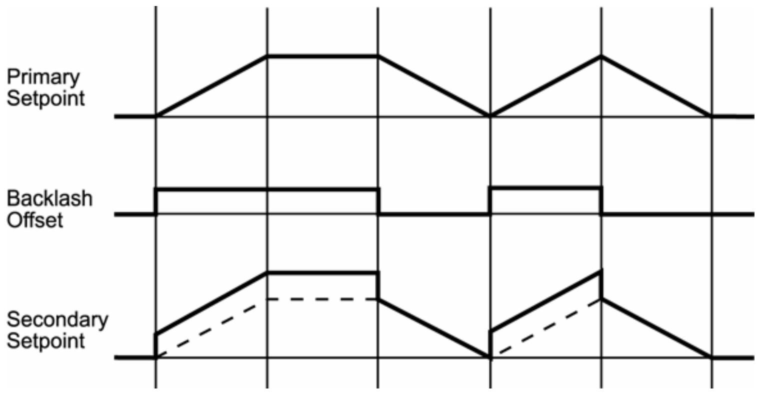
Format	BKL {*axis* {*value*}} {*axis* {*value*}} ...
Group	Setpoint Control
Units	Units scalable by PPU
Data Type	FP32
Default	0
Prompt Level	PROGx
See Also	AXIS, BSC, CAM, GEAR, HDW, JOG, PPU
Related Topics	Axis Parameters (0-7) (8-15)
Product Revision	Command and Firmware Release

This command sets or displays the backlash compensation of an axis. Backlash is primarily used to compensate for error introduced by hysteresis in mechanical gearboxes. Backlash is added to the secondary setpoint when the primary setpoint moves in the positive direction. If the primary setpoint is not changing, the backlash stays in its previous state. The backlash offset is used during the summation of the secondary setpoint.

This compensation is added in one servo update, therefore a large **BKL** offset will result in the motor "jerking" a little but the motion at the load should be smooth. For the same reason this feature might not be usable in a stepper application if the stepper translator cannot handle pulses too close together.

Issuing a **BKL** command to an axis without an argument will display the current setting for that axis. The default backlash is 0.0 for all axes.

The following illustrates backlash compensation:



NOTE: The primary setpoint is the summation of the current position and the total cam, gear, and jog offsets. The secondary setpoint is the summation of the primary setpoint and the total ballscrew and backlash offsets. The secondary setpoint is the one that is actually used by the servo loop.

Example

The following example sets backlash compensation for the X axis to 0.0025 units.

```
BKL X0.0025
```

BLK Turn On Block Mode

Format	BLK {PROG *number* | ALL}
Group	Program Control
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	AUT, PROG, STEP
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Turns on block mode for the currently selected program.

Arguments

PROG *number* | ALL

Remarks

Block mode is not applicable to PLC programs.

When you use **BLK** (block), the "block control" bit is set. If there is no master attached, the "block mode" bit is set as soon as the "block control" bit is detected. Otherwise, the program will feedhold, and then set the "block mode" when the master "in feedhold" is detected.

While in block mode, the program will use the **DEC** setting as the **STP** for all moves. This prevents consecutive moves with **STP 0** from coming to abrupt stops. When the program is taken out of block mode with the **AUT** command, moves operate normally.

Master "cycle start requests" are ignored until after the first **STEP** is issued in block mode.

The **BLK PROG** command will turn on block mode for the corresponding program and the **BLK ALL** command will turn on block mode for all programs. These commands can be issued from anywhere in the system.

Example

From the Terminal

The following turns on block mode in program 0:

```
P00> BLK
```

The following turns on block mode in program 0 from the system prompt:

```
SYS> PROG0
P00> BLK
```

The following turns on block mode in program 0 while remaining in program 1:

```
SYS> PROG1
P01> BLK PROG0
```

In a Program

The following is a subroutine that activates the **BLK** command to halt automation, then steps through the next three lines of code, then reactivates the automation:

```
_Blocksub
BLK
STEP
STEP
STEP
AUT
END
```

Example Bits

Flag Parameter 4128 For Program 0	
Block Control	1040
Block Mode	1042

Flag Parameter 4112 For Program 0	
In Feedhold	519

BLM Set Stroke Limit 'B'

Format	BLM {*axis* {*value*}} {*axis* {(*high*, *low*)}} ...
Group	Axis Limits
Units	Units scalable by PPU
Data Type	FP32
Default	0
Prompt Level	PROGx
See Also	ALM, AXIS, PPU
Related Topics	Axis Flags (0-7)(8-15), Secondary Axis Flags (0-7) (8-15), Axis Parameters (0-7) (8-15), Master Flags (0-7) (8-15)
Product Revision	Command and Firmware Release

This command sets the command position (current Position) limits monitored by the "not B limit" flags. When the command position of a given axis is outside of these limits, the appropriate flag is set. Otherwise, the flag is cleared. For masters, the flag is set if any of its slaves are outside of their limits—this is the only reaction the controller automatically performs. It is your responsibility in designing a program to react to these flags appropriately.

Issuing the **BLM** command to an axis without an argument displays the current positive and negative limits for that axis. Issuing the command with a single argument sets the positive limit to "value" and the negative limit to minus "value". Issuing the command with two arguments sets the positive limit to "high" and the negative limit to "low". The default for both is 0.0 for all axes.

The following is a table of 'Not B limit' flags:

MASTER	BIT
0	531
1	563
2	595
3	627
4	659
5	691
6	723
7	755

AXIS	BIT
0	771
1	803
2	835
3	867
4	899
5	931
6	963
7	995

Example 1

The following example sets the B limits to ±10 units for the X, Y and Z axes. If the axes are ever all within their B limits at the same time, the appropriate master flag will clear.

```
BLM X10 Y10 Z10
```

Example 2

The following example sets the B limits to +30 and –20 units for the X axis.

```
BLM X(30,-20)
```

BREAK Exit a Program Loop

Format	BREAK
Group	Program Flow
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	WHILE/WEND, FOR/TO/STEP/NEXT
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Allows termination of a **FOR/TO/STEP/NEXT** or **WHILE/WEND** loop under certain conditions.

Remarks

Conditions allowing the termination are:

- **FOR/TO/STEP/NEXT:** Terminates the **FOR** loop before reaching the **TO** value. The program continues with the command following the **NEXT** command.
- **WHILE/WEND:** Terminates the **WHILE** loop before the Boolean expression is false. The program continues with the command following the **WEND** command.

Example

```
#DEFINE Counter LV0
#DEFINE Stop BIT32

PROGRAM
DIM LV(2)

CLR Stop

FOR Counter=0 TO 10 STEP 2
     PRINT "Counting "
     PRINT "Seconds = ", Counter
     IF (STOP)
          PRINT " Breaking out of For Loop"
          BREAK
     ENDIF
     PRINT " DWELL FOR 2 SEC"
     DWL 2
NEXT

CLR Stop
COUNTER =0

WHILE (Counter < 10)
     PRINT "COUNTING "
     PRINT "SECONDDS = ", Counter
     IF (Stop)
          PRINT " Breaking out of While Loop"
          BREAK
```

```
    ENDIF
    PRINT " Dwell for 1 second"
    DWL 1
    Counter = Counter+1
WEND
ENDP
```

BRESET Disable Battery Backup

Format	BRESET
Group	Nonvolatile
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	ELOAD, ESAVE, ERASE, PBOOT
Related Topics	N/A
Product Revision	Command and Firmware Release

NOTE: This command does not apply to the ACR1500 and ACR1505 as the controllers do not have battery backup.

ACR8000

This command disables the battery backup the next time power is removed from the controller. This allows ACR8000 boards to be stored on the shelf without needlessly draining power from the battery. The next time power is applied to the controller, after shutting down with **BRESET** in effect, the battery will return to normal and will hold programs during consecutive power sequences.

NOTE: Once this command is issued, there is no way to return the battery to normal operation without removing and then restoring power. Stored programs will be lost.

ACR1200, ACR2000, and ACR8010

This command sets the battery backup memory to its default state the next time power is removed from the controller or the controller is reset. This allows the ACR1200 / ACR2000 / ACR8010 battery backed up memory to be cleared to default without physically removing the battery jumpers on the ACR1200 controller, the ACR8010 controller or the ACR2000 ACRCOMM controller. Reference the ACR1200, ACR2000, ACR8010 Hardware Manual (ACR2000 ACRCOMM section) for jumper settings to allow the boards to be stored on the shelf without needlessly draining power from the battery. The next time power is applied to the controller, after shutting down or after resetting the controller with **BRESET** in effect, the battery backed up memory will return to normal and will hold programs during consecutive power sequences.

If valid program data has been stored into the Flash, using the **FLASH SAVE** or **FLASH IMAGE** commands, this will overwrite the default conditions of the battery backed up memory when using **BRESET**. If the default memory conditions are required, a **FLASH ERASE** should be performed before the **BRESET** command.

NOTE: Once this command is issued, there is no way to return the battery backed up memory to normal operation without removing and then restoring power or resetting the controller. Stored programs will be lost.

Example

This command performs no function.

```
BRESET
```

BSC Ballscrew Compensation

Format	BSC *command* {*axis* {*data*}} {*axis* {*data*}} ...
Group	Setpoint Control
Units	LONG
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	AXIS, BKL, CAM, DIM, GEAR, HDW, JOG
Related Topics	Axis Flags (0-7)(8-15),
Product Revision	Command and Firmware Release

This command is used along with a second command to initialize and control ballscrew compensation for an axis. Ballscrew compensation is primarily used to compensate for nonlinear position error introduced by mechanical ballscrews. Ballscrew commands are identical to cam commands. Both ballscrews and cams can be active at the same time, each with different settings and offset tables.

The following is a list of valid ballscrew command combinations:

BSC DIM: Allocate ballscrew segments.

BSC FLZ: Set ballscrew input offset.

BSC ON: Enable ballscrew output.

BSC OFF: Disable ballscrew output.

BSC OFFSET: Set ballscrew output offset.

BSC RES: Transfer ballscrew offset.

BSC SCALE: Set ballscrew output scaling.

BSC SEG: Define ballscrew segment.

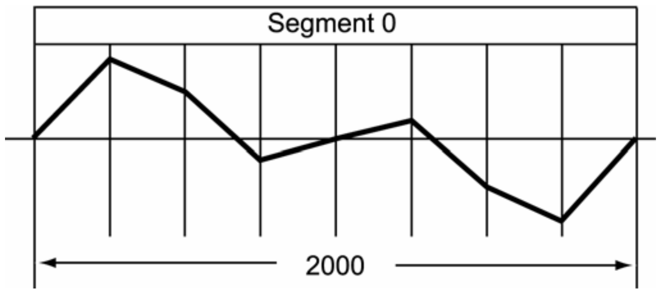
BSC SHIFT: Set incremental ballscrew shift.

BSC SRC: Redefine ballscrew source.

Ballscrew compensation, by default, uses the primary setpoint as its input source. Using the primary setpoint prevents feeding ballscrew offsets into the calculation of the ballscrew index and causing unstable conditions. Therefore, the **BSC SRC** command is not usually required.

NOTE: The primary setpoint is the summation of the current position and the total cam, gear, and jog offsets. The secondary setpoint is the summation of the primary setpoint and the total ballscrew and backlash offsets. The secondary setpoint is the one that is actually used by the servo loop.

The following illustrates a ballscrew table:



NOTE: The ballscrew compensation segment information is specific to the ballscrew in use. For more information, contact the mechanics manufacturer.

Example

The following example enables the X axis to use the above ballscrew table:

```
PROG0>
DIM LA(2)
DIM LA0(9)
LA0(00)=0
LA0(01)=853
LA0(02)=500
LA0(03)=-146
LA0(04)=0
LA0(05)=146
LA0(06)=-500
LA0(07)=-853
LA0(08)=0
BSC DIM X1
BSC SEG X(0,2000,LA0)
BSC ON X
```

BSC DIM   Allocate BSC Segments

Format	BSC DIM {*axis segments*} {*axis segments*} ...
Group	Setpoint Control
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	AXIS, BKL, BSC, CLEAR, DIM, GEAR, HDW, JOG
Related Topics	N/A
Product Revision	Command and Firmware Release

This command allocates working space for a ballscrew. The cam must be dimensioned before it can be initialized. This is in addition to the dimensioning done for the actual arrays attached to the cam segments. Newly dimensioned cams have no source defined for them, ballscrews point to the primary setpoint by default.

A cam can be composed of more than one segment with each segment having different distances between table entries. This allows some parts of the table to be defined coarsely and others to be defined in more detail.

The memory allocated by the **BSC DIM** command is a base of 52 bytes of working space plus an additional 24 bytes per defined segment.

Once a cam has been allocated, it can not be re-dimensioned to a different size without first doing a **CLEAR** to erase all dimensioning. This will also de-allocate any dimensioned user variables or cams. Do not allocate any more segments than are required by the application.

Example

The following example allocates two cam segments for the X axis and a single segment for the Y axis:

```
BSC DIM X2 Y1
```

BSC FLZ Set Ballscrew Input Offset

Format	BSC FLZ {*axis* {*offset*}} {*axis* {*offset*}} ...
Group	Setpoint Control
Units	input units
Data Type	Long
Default	0
Prompt Level	PROGx
See Also	AXIS, BKL, BSC, CAM, GEAR, HDW, JOG, SRC
Related Topics	N/A
Product Revision	Command and Firmware Release

This command sets or displays the ballscrew input offset of an axis. The ballscrew input offset is added to the ballscrew table index before it is used to calculate the actual table index. This is used to shift the zero of the table to the location of the input offset.

Issuing a **BSC FLZ** command to an axis without an argument will display the current setting for that axis. An error will be returned if the ballscrew has not been allocated with the **BSC DIM** command. The default ballscrew input offset is 0.0 for all axes.

Example

The following example shifts the X axis ballscrew table index by 250 units:

```
BSC FLZ X250
```

BSC OFF Disable Ballscrew Output

Format	BSC OFF {*axis*} {*axis*} ...
Group	Setpoint Control
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	AXIS, BKL, BSC, GEAR, HDW, JOG
Related Topics	N/A
Product Revision	Command and Firmware Release

This command disables ballscrew output for the designated axes. An error will be returned if the ballscrew has not been allocated with the **BSC DIM** command. Disabling the ballscrew immediately stops it.

Example

The following example disables the X and Y axis ballscrews:

```
BSC OFF X Y
```

BSC OFFSET Set Ballscrew Output Offset

Format	BSC OFFSET {*axis* {*scale*}} {*axis* {*scale*}} ...
Group	Setpoint Control
Units	output units scalable by PPU
Data Type	FP32
Default	0
Prompt Level	PROGx
See Also	AXIS, BKL, BSC, GEAR, HDW, JOG, PPU
Related Topics	N/A
Product Revision	Command and Firmware Release

This command sets or displays the ballscrew output offset of an axis. After the ballscrew table and index are used to interpolate an initial offset value, the value is multiplied by the ballscrew output scaling factor and then shifted by the ballscrew output offset. This number is then multiplied by the **PPU** of the ballscrew axis.

Issuing a **BSC OFFSET** command to an axis without an argument will display the current setting for that axis. An error will be returned if the ballscrew has not been allocated with the **BSC DIM** command. The default ballscrew output offset is 0.0 for all axes.

Example

The following example shifts the X axis ballscrew table output 500 units:

```
BSC OFFSET X500
```

BSC ON Enable Ballscrew Output

Format	BSC ON {*axis*} {*axis*} …
Group	Setpoint Control
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	AXIS, BKL, BSC, GEAR, HDW, JOG
Related Topics	N/A
Product Revision	Command and Firmware Release

This command enables ballscrew output for the designated axes. An error will be returned if the ballscrew has not been allocated with the **BSC DIM** command.

NOTE: Once **BSC** is enabled, it will stay enabled unless the "BSC cycles" parameter is set to a value other than zero.

Example

The following example enables the X, Y, and Z axis ballscrews:

```
BSC ON X Y Z
```

BSC RES Transfer Ballscrew Offset

Format	BSC RES {*axis* {*offset*}} {*axis* {*offset*}} ...
Group	Setpoint Control
Units	Units scalable by ppu
Data Type	FP32
Default	0
Prompt Level	PROGx
See Also	AXIS, BKL, BSC, CAM, GEAR, HDW, JOG, PPU
Related Topics	N/A
Product Revision	Command and Firmware Release

This command either clears or preloads the ballscrew offset of a given axis and adds the difference to the current position. This command will also clear out any ballscrew shift that may have been built up by an incremental ballscrew. The default ***offset*** argument is zero. The current position and ballscrew offset are adjusted according to the following formula:

- current position = current position + ballscrew offset - offset
- ballscrew offset = offset

When a ballscrew is turned off, the offset remains in the ballscrew offset parameter. The **BSC RES** command can be used to transfer the offset into the current position where it can be used as part of a normal move.

Example

The following example transfers the X axis ballscrew offset into the current position:

```
BSC RES X
```

BSC SCALE Set Ballscrew Output Scaling

Format	BSC SCALE {*axis* {*scale*}} {*axis* {*scale*}} …
Group	Setpoint Control
Units	None
Data Type	FP32
Default	1.0
Prompt Level	PROGx
See Also	AXIS, BKL, BSC, CAM, GEAR, HDW, JOG, PPU
Related Topics	N/A
Product Revision	Command and Firmware Release

This command sets or displays the ballscrew output scaling of an axis. After the ballscrew table and index are used to interpolate an initial offset value, the value is multiplied by the ballscrew output scaling factor and then shifted by the ballscrew output offset. This number is then multiplied by the **PPU** of the ballscrew axis.

Issuing a **BSC SCALE** command to an axis without an argument will display the current setting for that axis. An error will be returned if the ballscrew has not been allocated with the **BSC DIM** command. The default ballscrew output scaling is 1.0 for all axes.

Example

The following example scales the X axis ballscrew offset by 50 percent:

```
BSC SCALE X0.5
```

BSC SEG Define Ballscrew Segment

Format	BSC SEG {*axis* (*segment*, *length*, *array_name*)} ...
Group	Setpoint Control

Units	segment= none
	length= input units
	array_name= none

Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	AXIS, BKL, BSC, CAM, CLEAR, DIM, GEAR, HDW, JOG, SRC
Related Topics	N/A
Product Revision	Command and Firmware Release

This command defines the segments that were allocated with the **BSC DIM** command. The "segment" is a number from 0 to segments-1 and indicates which segment is being defined. The *length* argument defines the total length of the given segment (given in **BSC SRC** units). The *array_name* is the name of the longint array where the data points are to be stored. An error will occur if the ballscrew has not been allocated with the **BSC DIM** command.

A ballscrew can be composed of more than one segment with each segment having different distances between table entries. This allows some parts of the table to be defined coarsely and others to be defined in more detail.

The following internal formulas are modified by the **BSC SEG** information:

- distance between entries = segment length/(number of table entries-1)
- total length of the ballscrew = sum of segment lengths

Note that this information can be altered while the ballscrew is enabled, allowing the replacement of segments or the changing segment lengths on the fly.

Example 1

The following example defines the segment 1 of the X axis ballscrew as being 10000 units long and pointing to longint array LA0 for its data:

```
BSC SEG X(1,10000,LA0)
```

Example 2

Issuing just **BSC SEG** command will display ballscrew segment data.

```
P00>BSC SEG X
Seg_0 (0, 100, 500, 500)
Seg_1 (500, 1500, 2000, 3000)
Seg_2 (3000, 2500, 1000, 500)
Seg_3 (500, 500, 100, 0)
```

BSC SHIFT Set Incremental Ballscrew Shift

Format	BSC SHIFT {*axis* {*offset*}} {*axis* {*offset*}} ...
Group	Setpoint Control
Units	Output units scalable by PPU
Data Type	Long
Default	0
Prompt Level	PROGx
See Also	AXIS, BKL, BSC, CAM, GEAR, HDW, JOG, PPU
Related Topics	N/A
Product Revision	Command and Firmware Release

This command sets the incremental ballscrew shift. The first entry of one ballscrew segment is normally equal to the last entry of the previous segment. In cases where this is not true, the ballscrew is considered incremental. The starting "shift" for all ballscrews is 0.0. Issuing a **BSC SHIFT** with no argument will display the current reading.

Whenever an incremental ballscrew crosses a segment boundary, the difference between the two entries is used to adjust the ballscrew shift. The ballscrew shift is added to the interpolated offset to generate the actual ballscrew offset. If the total of all segment boundary shifts is not equal to zero, the overall pattern will be offset by that amount each cycle. Crossing ballscrew segment boundaries backwards will also adjust the ballscrew shift.

Example

The following example clears the X axis ballscrew shift:

```
BSC SHIFT X0
```

BSC SRC Redefine Ballscrew Source

Format	BSC SRC *axis sourcedef* {*axis sourcedef*} ...
Group	Setpoint Control
Units	none
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	AXIS, BSC, SRC
Related Topics	N/A
Product Revision	Command and Firmware Release

This command specifies the source for the input of a ballscrew. See the **SRC** command for the definition of the *sourcedef* argument.

This command sets a pointer to a memory area that is used to generate an index into the ballscrew table. Ballscrews do not have a default source assigned to them, ballscrews point to the primary setpoint by default. An error will occur if the ballscrew has not been allocated with the **BSC DIM** command.

Example

The following example sets the source of the X axis to encoder 3 and the source of the Y axis to the current position of AXIS1 (note that the parameter P12544 is not enclosed in parentheses):

```
BSC SRC X3 Y P12544
```

CAM Electronic Cam

Format	CAM *command* {*axis* {*data*}} {*axis* {*data*}} ...
Group	Setpoint Control
Units	LONG
Data Type	N/A
Default	PROGx
Prompt Level	N/A
See Also	AXIS, BKL, BSC, DIM, GEAR, HDW, JOG
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Used along with a second command to control an electronic cam for an axis.

Remarks

An electronic cam is primarily used as a replacement for a mechanical cam. Ballscrew commands are identical to cam commands. Both ballscrews and cams can be active at the same time, each with different settings and offset tables.

The following is a list of valid cam command combinations:

CAM CLEAR: Clear the cam setting.

CAM DIM: Allocate cam segments.

CAM FLZ: Set cam input offset.

CAM OFF: Disable cam output.

CAM OFFSET: Set cam output offset.

CAM ON: Enable cam output.

CAM ON IHPOS: Enable CAM on source position.

CAM ON TRG: Enable **CAM ON** from externally triggered capture position.

CAM ON TRGP: Enable **CAM ON** from externally triggered parameter position.

CAM POFF: Disable CAM at the end of the current CAM cycle.

CAM RES: Transfer cam offset.

CAM SCALE: Set cam output scaling.

CAM SEG: Define cam segment.

CAM SHIFT: Set incremental cam shift.

CAM SRC: Redefine cam source.

CAM SRC RES: Reset the CAM source.

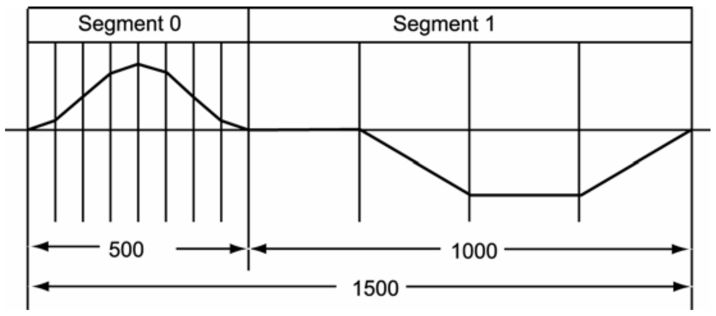
Cam uses an arbitrary source to generate an index into a table of offset values. If this index falls between two table entries, the cam offset is linearly interpolated between the entries. This offset is then scaled, shifted by the output offset, and then multiplied by the **PPU** for the given axis.

A cam table can be composed of more than one segment with each segment having different distances between table entries. The data for each segment of the table resides in separate longint arrays, possibly of different sizes. This allows some parts of the table to be defined coarsely and others to be defined in more detail. Each point of the cam table is scaled by PPU of the cam axis.

You can only use longint arrays in a cam table. The table index automatically tracks which segment it is in and where it is within that segment. It also wraps around if it goes off either end of the table. The wraparound point is determined by the total length of the table that is equal to the summation of the individual segment lengths.

NOTE: The primary setpoint is the summation of the current position and the total cam, gear, and jog offsets. The secondary setpoint is the summation of the primary setpoint and the total ballscrew and backlash offsets. The secondary setpoint is the one that is actually used by the servo loop.

The following illustrates a cam table (derived from In Example 1 below):



Examples

Example 1

The following example enables the X axis to use the above cam table with encoder number 4 as the input source:

```
PROG0>
DIM LA(2)     : REM Dimension 2 long arrays
DIM LA0(9)     : REM LAO has 9 elements
LA0(00)=0     : REM Start defining LAO cam table
LA0(01)=73
LA0(02)=250
LA0(03)=427
LA0(04)=500
LA0(05)=427
LA0(06)=250
LA0(07)=73
LA0(08)=0
DIM LA1(5)
LA1(00)=0
LA1(01)=0
LA1(02)=-500
LA1(03)=-500
LA1(04)=0
CAM DIM X2     : REM Define 2 cam segments
CAM SEG X(0,500,LA0)     : REM Define cam segment range and source
CAM SEG X(1,1000,LA1)
CAM SRC X4     : REM Define cam source as ENC4
CAM ON X     : REM Start camming
```

NOTE: The **CAM SRC** command must be issued after the cam segments have been defined. Improper operation may result from designating the **CAM SRC** first.

Example 2

Issuing just the **CAM** command displays the current setting of a cam, and can be used even if the cam is currently active. The example below shows the list

```
P00>CAM X
CAM FLZ X0
CAM OFFSET X0
CAM SCALE X1
CAM DIM X4
CAM SEG X (0, 2000, LA1)
CAM SEG X (1, 2000, LA3)
CAM SEG X (2, 2000, LA1)
CAM SEG X (3, 2000, LA0)
CAM SHIFT X0
CAM SRC X P12802
CAM ON X
```

Cam Cycles

The total length of the cam is the sum of segment lengths (see **CAM SEG**). In the graph above, the total length of the cam is 1500. One cam cycle is completed when the **CAM SRC** has traveled the total length of the cam, or 1500 counts in the illustration.

Cam Cycle Position and Cycle Number

The Cam Cycle Number is defined as the number of complete cycles represented by the change in SRC count. The Cam Cycle Position is defined as the number SRC counts within the current cycle. In Example 1 above, if the SRC count were 750, the Cam Cycle Number would be zero, and the Cam Cycle Position would be 750. But if the SRC count were 2000, the Cam Cycle Number would be one, and the Cam Cycle Position would be 500.

The Cam Cycle Number will go negative if the SRC count goes negative from the time the **CAM ON** is issued. The Cam Cycle Position will always be positive, and have a value between [0] and the [Cam Cycle Length *minus* 1]. In Example 1 above, if the SRC count were -250, the Cam Cycle Number would be -1, and the Cam Cycle Position would be 1250.

The controller automatically sets the Cam Cycle Number and Cam Cycle Position to zero with the **CAM ON** command and updates them as the SRC count changes. The Cam Cycle Position is present as SRC counts in Axis Parameter Index 0x7D, and the Cam Cycle Number is present in Axis Parameter Index 0x7E. The Cam Cycle Length is present in Axis Parameter Index 0x7F.

Axis parameter "Cam Cycles" can be set to run so many cam cycles. If this parameter is set to 3, then the cam will run for 3 times and then automatically turn itself off. The default value is zero, which means that the cam will run forever unless the user turns it off.

Cam Alignment

Cam Alignment refers to the starting position of the cam cycle within the cam table. If the cam cycle starts somewhere in the middle of the table, then a complete cam cycle will wrap around the end of the table and end where it started in the middle of the table. In most applications, it makes sense for the cam cycle to start at the beginning of the cam table. This will always be the case if the cam is started with the **CAM ON IHPOS**, **CAM ON TRG**, or **CAM ON TRGP** commands. But if a simple **CAM ON** command is used, alignment will depend on other factors.

Cam alignment begins with the **CAM SRC** and **CAM SRC RES** commands. The initial index into the cam table as a result of these commands is given with this formula:

```
    Index = (CAM SRC object count + CAM FLZ) % (total length of the cam)
```

This is the reason the **CAM SRC** command must be issued after the CAM segments are defined. In the case of the **CAM SRC RES** command, the **CAM SRC** count will be the supplied **RES** value. The index continues to track changes in **CAM SRC** count, even before the **CAM ON** command is issued, wrapping around the end of the table if necessary. If the cam is started with the **CAM ON IHPOS**, **CAM ON TRG**, or **CAM ON TRGP** commands, the index is set to zero, and will not track **CAM SRC** changes until the IHPOS, TRG, or TRGP conditions become true.

Two flags will modify the way the index changes with a simple **CAM ON** command. If neither is set, **CAM ON** will not change the index, so the cam offset will immediately jump to whatever table value is dictated by the current table index. If the “cam source zero” flag is set (Bit Index 6 in Secondary Axis Flags), the index is set to zero, and will then immediately track **CAM SRC** changes. This bit is automatically self-clearing with each **CAM ON** command, and so if this function is desired with each **CAM ON**, then the bit must be set before each **CAM ON**. If the “cam source FLZ” flag is set (Bit Index 0 in Axis Sixth Flags), the index is set to the current **CAM FLZ** value, and will then immediately track **CAM SRC** changes. This bit is not self-cleared and so only needs to be set once. If both bits are set, the “cam source FLZ” function overrides the “cam source zero” function.

The “cam source FLZ” flag offers the most convenient, predictable, and flexible control of cam table index when **CAM SRC** count is unknown or moving at the time of **CAM ON**.

More Examples

Example 3

The cam source is automatically reset when using the triggered cam. Each cam will run only three times.

```
P12400 = 3
CAM ON X TRG (0,0)     : REM Define cam to start when the rising primary
REM marker is seen
:
CAM ON X TRG (0,0)     : REM Define cam to start when the rising primary
REM marker is seen
```

Example 4

The cam source is explicitly reset and the source should not be moving before the cam is turned on otherwise the cam will not begin from the start.

```
P12400 = 3
CAM SRC X RES
CAM ON X     : REM The CAM will 3 times
CAM SRC X RES
CAM ON X     : REM The CAM will 3 times
```

CAM Velocity Smooth

Axis Parameter "CAM Velocity Smooth" is used to smooth out the CAM velocity that is used for the feed forward control. The Default value is 10, which means that the velocity is averaged on 10 samples to take away the jitter in the velocity term. The user can change this value to suite his application, however it should be changed before turning the **CAM ON**.

CAM Segment Parameter (Version 1.18.08 Update 13)

The "CAM Segment Active" parameter has been added to the Axis Parameters. It indicates the current, active segment of a CAM motion.

CAM CLEAR Clear the CAM Setting

Format	CAM CLEAR {*axis*}
Group	setpoint control
Units	LONG
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	AXIS, CAM
Related Topics	N/A
Product Revision	Command and Firmware Release

This command is used to clear the current setting of a cam. This command will turn off the cam and initialize all the cam variables to the default values. It can be used even if the cam is currently active.

NOTE: The memory allocated to the **CAM DIM** command will still be there and so will be the number of segments and their respective lengths. The cam can not be re-dimensioned by this command. If needed one has to use **CLEAR ALL** command to clear all the memory.

Example

The following example clears the cam settings on the Y axis.

```
CAM CLEAR Y
```

CAM DIM Allocate Cam Segments

Format	CAM DIM {*axis segments*} {*axis segments*} ...
Group	Setpoint Control
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	AXIS, BKL, BSC, CAM, CLEAR, DIM, GEAR, HDW, JOG
Related Topics	N/A
Product Revision	Command and Firmware Release

This command allocates working space for a cam. The cam must be dimensioned before it can be initialized. This is in addition to the dimensioning done for the actual arrays attached to the cam segments. Newly dimensioned cams have no source defined for them, ballscrews point to the primary setpoint by default.

A cam can be composed of more than one segment with each segment having different distances between table entries. This allows some parts of the table to be defined coarsely and others to be defined in more detail.

The memory allocated by the **CAM DIM** command is a base of 52 bytes of working space plus an additional 24 bytes per defined segment.

Once a cam has been allocated, it can not be re-dimensioned to a different size without first doing a **CLEAR** to erase all dimensioning. This will also de-allocate any dimensioned user variables or cams. Do not allocate any more segments than are required by the application.

Example

The following example allocates two cam segments for the X axis and a single segment for the Y axis:

```
CAM DIM X2 Y1
```

CAM FLZ Set Cam Input Offset

Format	CAM FLZ {*axis* {*offset*}} {*axis* {*offset*}} ...
Group	Setpoint Control
Units	input units
Data Type	Long
Default	0
Prompt Level	PROGx
See Also	AXIS, BKL, BSC, CAM, GEAR, HDW, JOG, SRC
Related Topics	N/A
Product Revision	Command and Firmware Release

This command sets or displays the cam input offset of an axis. The cam input offset is added to the cam table index before it is used to calculate the actual table index. This is used to shift the zero of the table to the location of the input offset.

Issuing a **CAM FLZ** command to an axis without an argument will display the current setting for that axis. An error will be returned if the cam has not been allocated with the **CAM DIM** command. The default cam input offset is 0.0 for all axes.

Example

The following example shifts the X axis cam table index by 250 units:

```
CAM FLZ X250
```

CAM OFF Disable Cam Output

Format	CAM OFF {*axis*} {*axis*} …
Group	Setpoint Control
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	AXIS, CAM, BKL, BSC, GEAR, HDW, JOG
Related Topics	N/A
Product Revision	Command and Firmware Release

This command disables cam output for the designated axes. An error will be returned if the cam has not been allocated with the **CAM DIM** command. Disabling the cam immediately stops it.

Example

The following example disables the X and Y axis cams:

```
CAM OFF X Y
```

CAM OFFSET Set Cam Output Offset

Format	CAM OFFSET {*axis* {*scale*}} {*axis* {*scale*}} ...
Group	Setpoint Control
Units	output units scalable by PPU
Data Type	FP32
Default	0
Prompt Level	PROGx
See Also	AXIS, BKL, BSC, CAM, GEAR, HDW, JOG, PPU
Related Topics	N/A
Product Revision	Command and Firmware Release

This command sets or displays the cam output offset of an axis. After the cam table and index are used to interpolate an initial offset value, the value is multiplied by the cam output scaling factor and then shifted by the cam output offset. This number is then multiplied by the **PPU** of the cammed axis.

Issuing a **CAM OFFSET** command to an axis without an argument will display the current setting for that axis. An error will be returned if the cam has not been allocated with the **CAM DIM** command. The default cam output offset is 0.0 for all axes.

Example

The following example shifts the X axis cam table output 500 units:

```
CAM OFFSET X500
```

CAM ON Enable Cam Output

Format	CAM ON {*axis*} {*axis*} ...
Group	Setpoint Control
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	AXIS, BKL, BSC, CAM, GEAR, HDW, JOG
Related Topics	N/A
Product Revision	Command and Firmware Release

This command enables cam output for the designated axes. An error will be returned if the cam has not been allocated with the **CAM DIM** command.

NOTE: Once **CAM** is enabled, it will stay enabled unless the "CAM cycles" parameter is set to a value other than zero.

Example

The following example enables the X, Y, and Z axis cams:

```
CAM ON X Y Z
```

CAM ON IHPOS Enable Cam on Source Position

Format 1	CAM ON {*axis*} IHPOS + {*setpoint*}...
Format 2	CAM ON {*axis*} IHPOS - {*setpoint*}
Format 3	AXIS{*axis*} CAM ON IHPOS + {*setpoint*}
Format 4	AXIS{*axis*} CAM ON IHPOS - {*setpoint*}
Group	Setpoint Control
Units	Pulses
Data Type	Long
Default	0
Prompt Level	SYS, PROGx
See Also	CAM
Related Topics	Tertiary Axis Flags (0-7) (8-15)
Product Revision	Command and Firmware Release

Description

This command arms the CAM to begin when the cam source becomes equal to and going greater then the setpoint value when using the + (default).

This command arms the CAM to begin when the cam source becomes equal to and going less then the setpoint value when using the –.

Remarks

The Cam Offset Register will be accurate to the setpoint value without any source counts loss.

The following discussion requires an understanding of the CAM SRC count value. The initial value of CAM SRC count is assigned during the **CAM SRC** or **CAM SRC RES** command, and takes on the same value as the index into the cam table. Please refer to the Cam Alignment section in the CAM command description. From then on, the CAM SRC count value continues to track changes in the value of the original CAM SRC object (for example, ENC0), but does not roll over at the end of the table, or get set to zero with a **CAM ON** command.

For example, suppose ENC0 has a value of 2250, the total cam table length is 1000, and the `CAM SRC ENC0` command is given. Now ENC0 is still 2250, and both the CAM SRC count and the cam table index are 250.

Now suppose ENC0 moves another 900 counts. Now ENC0 is 3150, CAM SRC count is 1150, and the cam table index is 150.

Now suppose the `CAM ON IHPOS 4150` command is given. Enc0 and CAM SRC count do not change, but the cam table index becomes zero.

By default, the CAM ON IHPOS setpoint is compared to the CAM SRC count, as described above. But if the CAM IHPOS SRC flag is set (Bit Index 1 in Axis Senary Flags), the CAM ON IHPOS setpoint is compared to the count of the object named in the **CAM SRC** command. In the example above, if the CAM IHPOS SRC flag is not set, the cam will become armed when CAM SRC count is 4150, which is 3000 counts away. But if the CAM IHPOS SRC flag is set, the cam will become armed when ENC0 count is

4150, only 1000 counts away. This bit is not self-cleared and so only needs to be set once.

This command does not inhibit program operation. Once this command is issued, the program will continue to process commands. The cam will start when the setpoint conditions are met. The **CAM ON IHPOS** command can be aborted with the **CAM OFF** command and clearing the appropriate CAM IHPOS flag.

In the Tertiary Axis Flags, you can use the CAM ON IHPOS Armed flag to clear **CAM ON IHPOS**.

NOTE: If the **CAM IHPOS** command is issued and the CAM source does not reach the setpoint, the cam may not function correctly unless the **CAM IHPOS** flag is cleared.

Examples

Usage Example

Issuing the following command will arm the CAM to be turn on as soon as the CAM source crosses the position of 5000 pulses.

```
CAM ON Y IHPOS (5000)
```

Similarly, the following command will arm the CAM. The CAM will not turn on till the CAM source crosses the position of 12345 pulses. At this cross over point the CAM will automatically turn on. Thus giving the ability to precisely turn on the CAM at a particular location.

```
AXIS0 CAM ON IHPOS (12345)
```

Sample Program

This is detail program running a CAM profile. The CAM will precisely turn at target position LV2. The value of LV2 is modified as required by the application. Since the CMA cycle is set to one. The CAM will execute one complete cycle and turn itself off.

```
ATTACH MASTER0
ATTACH SLAVE0 AXIS0 "X"
ATTACH SLAVE1 AXIS1 "Y"

MULT X4 Y4
PPU X1 Y1
F 300
ACC 1000
DEC 1000
STP 1000
CAM CLEAR Y
JOG VEL X 500

CAM DIM Y2
CAM SEG Y(0,500,LA0)
CAM SEG Y(1,1000,LA1)
REM - Cam Source set to Secondary Setpoint Register for Axis X
CAM SRC Y P12295
REM - Set Cam Cycles to 1
P12656 = 1
LV1 = 1
REM - Initial trigger point
LV2 = 1000
REM - Distance between trigger points
LV3 = 3000

JOG FWD X
_LOOP1
```

```
IF (LV1 = 10) THEN GOTO 500
PRINT "Trigger Point  = ";LV2

CAM ON Y IHPOS + (LV2)
PRINT "CAM cycle ";LV1;" Ready";
REM Wait for CAM cycle to be completed
INH -822
PRINT "  <== Done "
PRINT " "
REM - Set next trigger point
LV2 = (LV2 + LV3)
LV1 = (LV1 + 1)
GOTO LOOP1

_LOOP2
JOG OFF X
JOG RES X
RES X Y
PRINT "FINISHED"
END

DIM LV 5
DIM LA(2)
DIM LA0(9)
LA0(00) = 0
LA0(01) = 125
LA0(02) = 250
LA0(03) = 500
LA0(04) = 500
LA0(05) = 500
LA0(06) = 250
LA0(07) = 125
LA0(08) = 0
DIM LA1(5)
LA1(00) = 0
LA1(01) = 0
LA1(02) = -500
LA1(03) = -500
LA1(04) = 0
```

CAM ON TRG Enable External Source Trigger CAM

Format	CAM ON {*axis*} TRG (*mode*, *capture_register*)
Group	Setpoint Control
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	AXIS, BKL, BSC, CAM, GEAR, HDW, INTCAP, JOG
Related Topics	Secondary Axis Flags (0-7) (8-15)
Product Revision	Command and Firmware Release

NOTE: This command does not apply to the ACR8000 controller.

This command arms the loaded **CAM** to begin when an externally sourced trigger occurs. The latency error is 1 microsecond (400 nanoseconds for the ACR9000). The mode parameter and hardware capture register information for the **CAM ON TRG** is the same as those used in the **INTCAP** command.

For valid mode and capture registers, see INTCAP.

NOTE: It is recommended that the cam source be attached before the source starts to move.

Example

The following example enables or starts the Y axis cam when triggered by rising primary marker of encoder 0.

```
CAM SRC Y 0
CAM ON Y TRG(0,0)
```

If the cam needs to be turned off and armed again then issue the following commands

```
CAM OFF Y
CAM RES Y
CAM ON Y TRG(0,0)
```

The following example initiates the cam profile on the rising external trigger, and runs for 500 counts of the master axis.

```
PROG0>
ATTACH MASTER0
ATTACH SLAVE0 AXIS0 "X"
CLEAR    : REM Clears dimensioning of local variables
DIM LA(1)    : REM Dimension one long array
DIM LA0(3)    : REM Dimension LA0 for 3 elements
PPU X1    : REM Set scaling for axis 0
LA0 (0)=0    : REM Fill long array 0
LA0 (1)=73
LA0 (2)=500
CAM DIM X1    : REM Dimension on cam table
CAM SEG X (0,500,LA0)    : REM Assign cam segment to LA0
CAMSRC X2    : REM Set cam source to ENC2
CAM ON X TRG(2,0)    : REM Cam start on rising primary external
```

CAM ON TRGP Enable External Trigger CAM

Format	CAM ON {*axis*} TRGP (*mode*, *capture_register*)
Group	Setpoint Control
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	AXIS, BKL, BSC, CAM, GEAR, HDW, JOG, SRC
Related Topics	N/A
Product Revision	Command and Firmware Release

NOTE: This command does not apply to the ACR8000 controller.

This command is same as the **CAM ON TRG**, except that the cam source can be any P parameter. In this case the capture register value is not used since the cam source value is different than the capture register value. This is less precise than the **CAM ON TRG** and the worse case error can be one servo period.

For valid mode and capture registers, see INTCAP.

Example

The following example enables or starts the Y axis cam when triggered by rising primary marker of encoder 0.

```
CAM SRC Y P12288
CAM ON Y TRGP(0,0)
```

If the CAM needs to be turned off and armed again then issue the following commands

```
CAM OFF Y
CAM RES Y
CAM ON Y TRGP(0,0)
```

CAM POFF Disable CAM at End of Current CAM Cycle

Format 1	CAM POFF {*axis*}
Format 2	AXIS{*axis*} CAM POFF
Group	Setpoint Control
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx
See Also	CAM
Related Topics	Tertiary Axis Flags (0-7) (8-15)
Product Revision	Command and Firmware Release

This command arms the CAM to end when the current cam cycle is complete.

The Cam Offset Register will be accurate to the setpoint value without any source encoder counts loss.

This command does not inhibit program operation. Once this command is issued, the program will continue to process commands. The cam will continue until it reaches the end of the current cam cycle. The **CAM POFF** command can be aborted with the **CAM OFF** command and clearing the appropriate **CAM POFF** flag.

In the Tertiary Axis Flags, you can use the “CAM POFF Armed” flag to clear **CAM POFF**.

NOTE: If the **CAM POFF** command is issued and the CAM source encoder does not reach the end of the current cam cycle, the cam may not function correctly unless the **CAM POFF** flag is cleared.

Example

Issuing the following command will not immediately turn of the CAM, rather it will wait for the CAM cycle to complete and then the CAM will turn off exactly at the end of CAM cycle.

```
CAM POFF Y
```

This is an alternate way to issue the above command.

```
AXIS1 CAM POFF
```

Sample Program

Turn on cam using the **CAM ON Y** command. When the **CAM POFF Y** command is issued, the cam will continue until the end of the current cam cycle.

```
ATTACH MASTER0
ATTACH SLAVE0 AXIS0 "X"
ATTACH SLAVE1 AXIS1 "Y"

MULT X4 Y4
PPU X1 Y1
F 300
ACC 1000
DEC 1000
STP 1000
CAM CLEAR Y
```

```
JOG VEL X 500

CAM DIM Y2
CAM SEG Y(0,500,LA0)
CAM SEG Y(1,1000,LA1)
REM - Cam Source set to Secondary Setpoint Register for Axis X
CAM SRC Y P12295
REM - Set Cam Cycles continuous cycle
P12656 = 0
LV1 = 1
REM - Initial trigger point
LV2 = 1000
REM - Distance between trigger points
LV3 = 3000
JOG FWD X

DIM LV 5
DIM LA(2)
DIM LA0(9)
LA0(00) = 0
LA0(01) = 125
LA0(02) = 250
LA0(03) = 500
LA0(04) = 500
LA0(05) = 500
LA0(06) = 250
LA0(07) = 125
LA0(08) = 0
DIM LA1(5)
LA1(00) = 0
LA1(01) = 0
LA1(02) = -500
LA1(03) = -500
LA1(04) = 0
```

CAM RES Transfer Cam Offset

Format	CAM RES {*axis* {*offset*}} {*axis* {*offset*}} …
Group	Setpoint Control
Units	Units scalable by PPU
Data Type	FP32
Default	0
Prompt Level	PROGx
See Also	AXIS, BKL, BSC, CAM, GEAR, HDW, JOG, PPU
Related Topics	Axis Parameters (0-7) (8-15)
Product Revision	Command and Firmware Release

This command either clears or preloads the cam offset of a given axis and adds the difference to the current position. This command will also clear out any cam shift that may have been built up by an incremental cam. The default *offset* argument is zero. The current position and cam offset are adjusted according to the following formula:

- Current position = current position + cam offset - offset
- Cam offset = offset

When a cam is turned off, the offset remains in the cam offset parameter. The **CAM RES** command can be used to transfer the offset into the current position where it can be used as part of a normal move.

Example

The following example transfers the X axis cam offset into the current position:

```
CAM RES X
```

CAM SCALE Set Cam Output Scaling

Format	CAM SCALE {*axis* {*scale*}} {*axis* {*scale*}} ...
Group	Setpoint Control
Units	None
Data Type	FP32
Default	1.0
Prompt Level	PROGx
See Also	AXIS, BKL, BSC, CAM, GEAR, HDW, JOG, PPU
Related Topics	N/A
Product Revision	Command and Firmware Release

This command sets or displays the cam output scaling of an axis. After the cam table and index are used to interpolate an initial offset value, the value is multiplied by the cam output scaling factor and then shifted by the cam output offset. This number is then multiplied by the **PPU** of the cammed axis.

Issuing a **CAM SCALE** command to an axis without an argument will display the current setting for that axis. An error will be returned if the cam has not been allocated with the **CAM DIM** command. The default cam output scaling is 1.0 for all axes.

Example

The following example scales the X axis cam offset by 50 percent:

```
CAM SCALE X0.5
```

CAM SEG Define Cam Segment

Format	CAM SEG {*axis* (*segment*, *length*, *array_name*)} ...
Group	Setpoint Control

Units	segment= none
	length= input units
	array_name= none

Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	AXIS, BKL, BSC, CAM, CLEAR, DIM, GEAR, HDW, JOG, SRC
Related Topics	N/A
Product Revision	Command and Firmware Release

This command defines the segments that were allocated with the **CAM DIM** command. The "segment" is a number from 0 to segments-1 and indicates which segment is being defined. The *length* argument defines the total length of the given segment (given in **CAM SRC** units). The *array_name* is the name of the longint array where the data points are to be stored. An error will occur if the cam has not been allocated with the **CAM DIM** command.

A cam can be composed of more than one segment with each segment having different distances between table entries. This allows some parts of the table to be defined coarsely and others to be defined in more detail.

The following internal formulas are modified by the **CAM SEG** information:

- distance between entries = segment length/(number of table entries-1)
- total length of the cam = sum of segment lengths

Note that this information can be altered while the cam is enabled, allowing the replacement of segments or the changing segment lengths on the fly.

Example 1

The following example defines the segment 1 of the X axis cam as being 10000 units long and pointing to longint array LA0 for its data:

```
CAM SEG X(1,10000,LA0)
```

Example 2

Issuing just **CAM SEG** command will display cam segment data.

```
P00>CAM SEG X
Seg_0 (0, 100, 500, 500)
Seg_1 (500, 1500, 2000, 3000)
Seg_2 (3000, 2500, 1000, 500)
Seg_3 (500, 500, 100, 0)
```

CAM SHIFT Set Incremental Cam Shift

Format	CAM SHIFT {*axis* {*offset*}} {*axis* {*offset*}} ...
Group	Setpoint Control
Units	Output units scalable by PPU
Data Type	Long
Default	0
Prompt Level	PROGx
See Also	AXIS, BKL, BSC, CAM, GEAR, HDW, JOG, PPU
Related Topics	N/A
Product Revision	Command and Firmware Release

This command sets the incremental cam shift. The first entry of one cam segment is normally equal to the last entry of the previous segment. In cases where this is not true, the cam is considered incremental. The starting "shift" for all cams is 0.0. Issuing a **CAM SHIFT** with no argument will display the current reading.

Whenever an incremental cam crosses a segment boundary, the difference between the two entries is used to adjust the cam shift. The cam shift is added to the interpolated offset to generate the actual cam offset. If the total of all segment boundary shifts is not equal to zero, the overall pattern will be offset by that amount each cycle. Crossing cam segment boundaries backwards will also adjust the cam shift.

Example

The following example clears the X axis cam shift:

```
CAM SHIFT X0
```

CAM SRC Redefine Cam Source

Format	CAM SRC *axis sourcedef* {*axis sourcedef*} ...
Group	Setpoint Control
Units	none
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	AXIS, CAM, SRC
Related Topics	N/A
Product Revision	Command and Firmware Release

This command specifies the source for the input of a cam. See the SRC command for the definition of the *sourcedef* argument.

This command sets a pointer to a memory area that is used to generate an index into the cam table. Cams do not have a default source assigned to them, ballscrews point to the primary setpoint by default. An error will occur if the cam has not been allocated with the **CAM DIM** command.

Example

The following example sets the source of the X axis to encoder 3 and the source of the Y axis to the current position of AXIS1 (note that the parameter P12544 is not enclosed in parentheses):

```
CAM SRC X3 Y P12544
```

CAM SRC RES Reset the Cam Source

Format	CAM SRC {*axis*} RES
Group	Setpoint Control
Units	Units scalable by PPU
Data Type	FP32
Default	0
Prompt Level	PROGx
See Also	AXIS, PPU, SRC
Related Topics	N/A
Product Revision	Command and Firmware Release

This command resets the source for the input of a cam. Usually it will be used when the cam is off and the source needs to be reset to a certain value.

Example

```
CAM SRC X RES    : REM Reset source of X to zero
CAM SRC X RES 100    : REM Reset source to 100
CAM SHIFT X0
```

CIP Ethernet/IP Status

Format	CIP
Group	Global Objects
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	N/A
Related Topics	N/A
Product Revision	Command and Firmware Release

The **CIP** command displays the status of the Ethernet/IP connection to the controller. There are two modes of communication, Class 1 I/O (UDP) and Class 3 messages (TCP).

Example 1

When no device is connected to the controller over Ethernet/IP, the controller reports the following: No class one or class three streams are connected; no class one or class three messages have been received; there are no class three messages in queue; and no devices are connected.

```
SYS>CIP
Class1 I/O Stream = 0
Class3 Message Stream = 0
Class1 Received = 0
Class3 Received = 0
Messages Queued = 0
Total number of connections = 0
First Client IP activating CIP "0.0.0.0"
```

Example 2

When a client device connects to the controller over Ethernet/IP, the controller provides a report of which streams are connected and what has occurred.

```
SYS>CIP
Class1 I/O Stream = 2
Class3 Message Stream = 3
Class1 Received = 1109
Class3 Received = 53
Messages Queued = 1
Total number of connections = 2
First Client IP activating CIP "172.25.8.22"
```

CIRCCW 2-Dimensional Counter Clockwise Circle

Format	CIRCCW *axis* (*target*, *center*) *axis* (*target*, *center*)
Group	Interpolation
Units	Units scalable by PPU
Data Type	FP32
Default	N/A
Prompt Level	PROGx
Report Back	None **To view, type** N/A
See Also	CIRCW, SINE, PPU
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Generates a circular profile on a plane defined by two axes.

Arguments

axis

Required. Assigns an axis.

target

Required. Position at the end of the move (in units).

center

Required. Center of the circle being drawn (in units).

Remarks

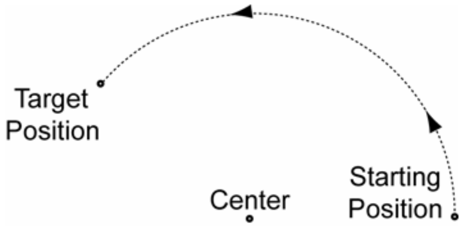
The current position of the axes is the starting position for circular motion. Using the starting position and the ***target*** and ***center*** arguments, the coordinated motion profiler calculates the circular motion using the following formulas:

theta = start angle of arc

radius = radius of the arc = target - center

xstart = xcenter + radius * cos(theta)

ystart = ycenter + radius * sin(theta)



Caution — Be sure to move the axes to the correct starting point before the move begins. If the axes are not at the correct radius from the center point, the axes jump to a position calculated by the coordinated motion profiler.

The coordinated motion profiler uses the center point, calculated radius, and the current position to determine the starting point. The current position and center point are viewed as creating a radial line, and the starting point is fixed at the calculated radius.

Examples

Example 1

Syntax examples.

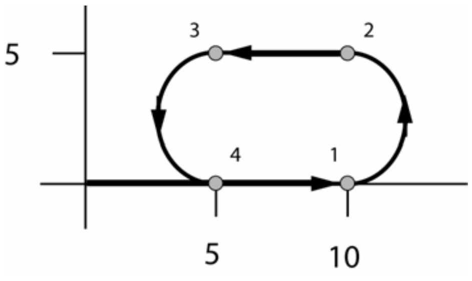
```
REM Variable target and center
CIRCCW X ((LV1),(LV2)) Y ((LV3),(LV4))
REM Absolute target and center
CIRCCW X (59.078, -59) Y (-75.576, -100)
REM X-axis has incremental target and center
CIRCCW X/(110.23, 134) Y (-152.309, -80)
```

Example 2

Elongated hole.

```
ACC 5 VEL 1 DEC 5 STP 5 JRK 25

X0 Y0
X10   : REM linear move to point 1 at (10,0)
CIRCCW X(10,10) Y(5,2.5)    : REM arc move to point 2 at (10,5)
REM This arc’s center is (10,2.5)
X5 Y5   : REM linear move to point 3 at (5,5)
CIRCCW X(5,5) Y(0,2.5)    : REM arc move to point 4 at (5,0)
REM this arc’s center is at (5,2.5)
```

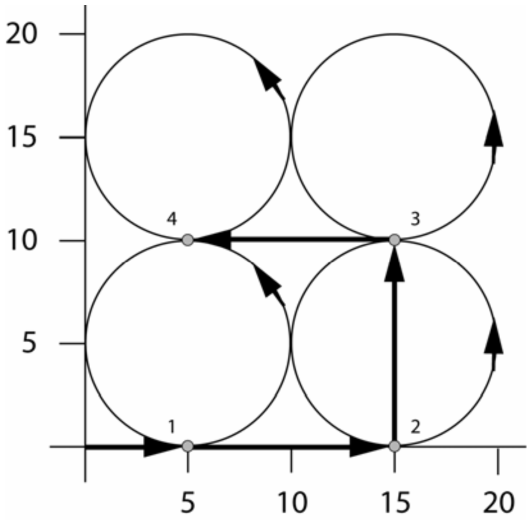


Example 3

An XY stage is used to draw 4 identical circles at different locations. Variables and incremental targets and centers are used in this example.

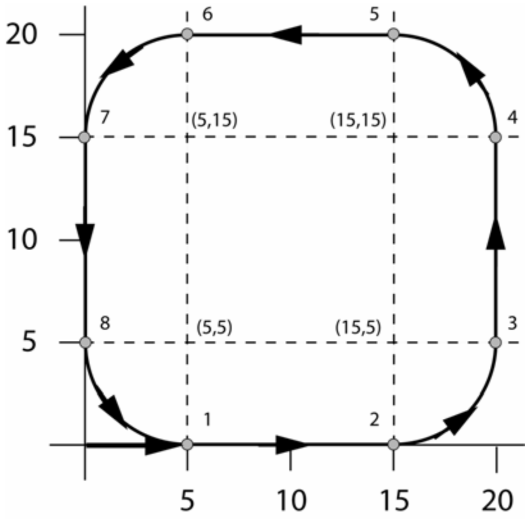
```
PROGRAM
DIM SV(1)
ACC 5 VEL 1 DEC 5 STP 5 JRK 25
SV0=5.0    : REM desired circle radius assigned to local variable
X0 Y0   : REM move to zero location
X/(SV0) : REM linear move to point 1
CIRCCW X/(0,0) Y/(0,SV0)
REM Arc move for a full circle back to starting point 1
X/(SV0*2)    : REM linear move to point 2
CIRCCW X/(0,0) Y/(0,SV0)
REM Arc move for a full circle back to starting point 2
Y/(SV0*2)    : REM vertical move to point 3
CIRCCW X/(0,0) Y/(0,SV0)
REM Arc move for a full circle back to starting point 3
X/(-SV0*2)    : REM horizontal linear move to point 4
```

```
CIRCCW X/(0,0) Y/(0,SV0)
REM Arc move for a full circle back to starting point 4
ENDP
```



Example 4

```
PROGRAM
ACC 5 VEL 1 DEC 5 STP 5 JRK 25
X0 Y0
STP0   : REM  stop ramp is deactivated
REM Moves will blend together at constant vector velocity
X5   : REM linear move to point 1
X15  : REM linear move to point 2
CIRCCW X(20,15) Y(5,5)  : REM arc move to point 3
Y15  : REM linear move to point 4
CIRCCW X(15,15) Y(20,15)  : REM arc move to point 5
X5  : REM linear move to point 6
CIRCCW X(0,5) Y(15,15)  : REM arc move to point 7
Y5  : REM linear move to point 8
STP 5   : REM stop ramp is activated for final move
CIRCCW X(5,5) Y(0,5)  : REM arc move to 1
ENDP
```



CIRCW 2-Dimensional Clockwise Circle

Format	CIRCW *axis* (*target*, *center*) *axis* (*target*, *center*)		
Group	Interpolation		
Units	Units scalable by PPU		
Data Type	FP32		
Default	N/A		
Prompt Level	PROGx		
Report Back	None	**To view, type**	N/A
See Also	CIRCCW, SINE, PPU		
Related Topics	N/A		
Product Revision	Command and Firmware Release		

Description

Generates a circular profile on a plane defined by two axes.

Arguments

axis

Required. Assigns an axis.

target

Required. Position at the end of the move (in units).

center

Required. Center of the circle being drawn (in units).

Remarks

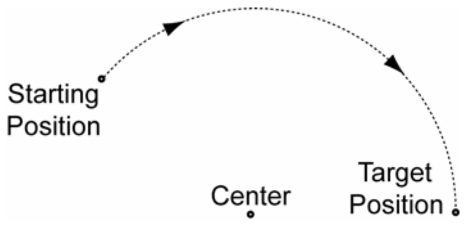
The current position of the axes is the starting position for circular motion. Using the starting position and the *target* and *center* arguments, the coordinated motion profiler calculates the circular motion using the following formulas:

theta = start angle of arc

radius = radius of the arc = target - center

xstart = xcenter + radius * cos(theta)

ystart = ycenter + radius * sin(theta)



Caution — Be sure to move the axes to the correct starting point before the move begins. If the axes are not at the correct radius from the center point, the axes jump to a position calculated by the coordinated motion profiler.

The coordinated motion profiler uses the center point, calculated radius, and the current position to determine the starting point. The current position and center point are viewed as creating a radial line, and the starting point is fixed at the calculated radius.

Examples

Example 1

```
CIRCW X ((LV1),(LV2)) Y ((LV3),LV4)) : REM Variable target & center
CIRCW X (59.078, -59) Y (-75.576, -100) : REM Absolute target & center
CIRCW X/(110.23, 134) Y (-152.309, -80)
REM X-axis has incremental target and center
```

See **CIRCCW** for additional examples.

CLEAR Clear Memory Allocation

Format	CLEAR
Group	Memory Control
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx
See Also	DIM, MEM
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Frees memory that was dimensioned for programs, variables, arrays, and data logging.

Remarks

The **CLEAR** command behaves specific to the system level or program level:

- At the system level, the **CLEAR** command frees memory allocated to all programs. To do this, the programs must be empty. If the programs are not empty, the controller generates an error, "Programs not empty."

  You can also return stream buffers to their default storage (256 bytes). When the resizing of a stream buffer is complete, the controller then sets the "Re-dimensioned" flag (bit 13) in the Stream Flags.

- At the program level or from within a running program, the **CLEAR** command frees memory allocated to local variables and arrays.

NOTE: After clearing, use the **DIM** command to reallocate memory as required.

The following is a list of valid command combinations:

CLEAR DPCB	Erases the DPCB buffer.
CLEAR FIFO	Erases the FIFO buffer.
CLEAR COM1	Erases the COM1 buffer.
CLEAR COM2	Erases the COM2 buffer.
CLEAR STREAM	Erases the STREAM buffer.

Example

The following prepares the controller for a factory reset. It stops all motion, clears all programs and attachments, and then removes all memory dimensioning.

```
SYS
HALT ALL
NEW ALL
DETACH ALL
CLEAR
```

The following illustrates using DIM to erase local memory dimensioning.

```
SYS>DIM PROG0(10000)
SYS>PROG0
P00>DIM LV(100)
P00>DIM
DIM LV(100)
POO>CLEAR
P00>DIM
P00>
```

CLEAR COM1 Clear Memory Allocation

Format	CLEAR COM1
Group	Memory Control
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS
See Also	DIM, MEM
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Clears memory allocation for COM1 stream buffers.

Remarks

When you use the **CLEAR COM1** command, the memory is returned to the default storage allocation (256 bytes). You can then reallocate the memory storage using the **DIM** command.

When the resizing of a stream buffer is complete, the controller then sets the “Re-dimensioned” flag (bit 13) in the Stream Flags.

Example

The following removes all memory dimensioning for COM1 stream buffers.

```
SYS
CLEAR COM1
```

CLEAR COM2 Clear Memory Allocation

Format	CLEAR COM2
Group	Memory Control
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS
See Also	DIM, MEM
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Clears memory allocation for COM2 stream buffers.

Remarks

When you use the **CLEAR COM2** command, the memory is returned to the default storage allocation (256 bytes). You can then reallocate the memory storage using the **DIM** command.

When the resizing of a stream buffer is complete, the controller then sets the "Re-dimensioned" flag (bit 13) in the Stream Flags.

Example

The following removes all memory dimensioning for stream buffers.

```
SYS
CLEAR COM2
```

CLEAR DPCB Clear Memory Allocation (ACR8020 and ACR1505)

Format	CLEAR DPCB
Group	Memory Control
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS
See Also	DIM, MEM
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Clears memory allocation for dual port communication bus (DPCB) stream buffers.

Remarks

When you use the **CLEAR DPCB** command, the memory is returned to the default storage allocation (256 bytes). You can then reallocate the memory storage using the **DIM** command.

When the resizing of a stream buffer is complete, the controller then sets the “Re-dimensioned” flag (bit 13) in the Stream Flags.

Example

The following removes all memory dimensioning for DPCB stream buffers.

```
SYS
CLEAR DPCB
```

CLEAR FIFO Clear Memory Allocation (ACR1500, ACR2000, ACR8010)

Format	CLEAR FIFO
Group	Memory Control
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS
See Also	DIM, MEM
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Clears memory allocation for first in, first out (FIFO) stream buffers.

Remarks

When you use the **CLEAR FIFO** command, the memory is returned to the default storage allocation (256 bytes). You can then reallocate the memory storage using the **DIM** command.

When the resizing of a stream buffer is complete, the controller then sets the "Re-dimensioned" flag (bit 13) in the Stream Flags.

Example

The following removes all memory dimensioning for FIFO stream buffers.

```
SYS
CLEAR FIFO
```

CLEAR STREAM Clear Memory Allocation

Format	CLEAR STREAM *stream*
Group	Memory Control
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	DIM, MEM
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Clears memory allocation for stream buffers.

Arguments

stream1	Stream for USB port
stream2 - stream5	Streams assigned to Ethernet ports

Remarks

When you use the **CLEAR STREAM** command, the memory is returned to the default storage allocation (256 bytes). You can then reallocate the memory storage using the **DIM** command.

When the resizing of a stream buffer is complete, the controller then sets the “Re-dimensioned” flag (bit 13) in the Stream Flags.

Example

The following removes all memory dimensioning for stream buffers.

```
SYS
CLEAR STREAM1
CLEAR STREAM2
CLEAR STREAM3
CLEAR STREAM4
CLEAR STREAM5
```

CLOSE Close a Device

Format	CLOSE #*device*
Group	Character I/O
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	INPUT, OPEN, PRINT
Related Topics	N/A
Product Revision	Command and Firmware Release

This command closes a device. The valid range for ***device*** is zero through three. Each program has its own device zero which is used as its default device. Devices one through three are controller-wide system resources that can be closed from within any program or from any system or program prompt.

When a device is opened, the operating system attached to that device enters an idle state, allowing incoming characters to be used by a program instead of being interpreted as commands. When the device is closed, the device will enter its auto-detect mode as if it were starting from power-up.

Example

```
CLOSE #1
```

CLR Clear a Bit Flag

Format	CLR *index*
Group	Logic Function
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	BIT, SET, INH
Related Topics	N/A
Product Revision	Command and Firmware Release

This command clears the specified bit flag. This flag can either be a physical output or an internal bit flag.

Example

The following example will clear output 32:

```
CLR 32
```

CONFIG Hardware Configuration

Format	CONFIG {*command* | *configlist*}
Group	Operating System
Units	N/A
Data Type	N/A
Default	See below
Prompt Level	SYS, PROGx, PLCx
See Also	AXIS, FLASH
Related Topics	N/A
Product Revision	Command and Firmware Release

This command defines the base hardware installed on the boards, including the encoders and the hardware modules installed in the SIMM sockets. The command also allows onboard and expansion IO to be redirected for the ACR1200, ACR1505, ACR2000, ACR8000, ACR8010, and ACR8020 boards.

Issuing a **CONFIG** command without any arguments will display the current configuration. If the IO/XIO have not been redirected, their configurations will not be displayed (IO/XIO not available on ACR1500). For the ACR1500, **CONFIG IO MODE**, **CONFIG IO OUT**, and **CONFIG IO INPUT** configurations will be displayed.

Hardware configurations that have been set by the user with the **CONFIG** commands will be automatically saved in the EEPROM/FLASH by the processor. FLASH and/or EEPROM commands (**FLASH SAVE**, **FLASH ERASE**, **ERASE**, **ESAVE**, etc.) have no effect on the saved hardware configuration information.

The following is a list of valid **CONFIG** command combinations:

CONFIG *configlist*: Set up hardware configuration (axes 0-7).

CONFIG CLEAR: Reset the default configuration.

CONFIG IO: Redirect the onboard I/O.

CONFIG IO INPUT: Configure the polarity of the input logic.

CONFIG IO MODE: Set the onboard 82C55 I/O mode.

CONFIG IO OUT: Configure the polarity of the output logic.

CONFIG XAXIS *configlist*: Set up hardware configuration (axes 8-15) (ACR1505 and ACR8020)

The following is the syntax of the *configlist* argument:

```
Configlist = encoders module0 module1 module2
Encoders  = NONE | ENC2 | ENC3¹ | ENC4 | ENC5¹ | ENC6 | ENC8 | ENC10¹
module0  = NONE | DAC2 | DAC4 | STEPPER2 | STEPPER4 | DACSTEP2² | DACSTEP4⁴
module1  = NONE | DAC2 | DAC4 | STEPPER2 | STEPPER4 | DACSTEP2² | DACSTEP4³
module2  = NONE | ADC
```

1. ENC3 is valid only for the ACR1200 controller. ENC5 and ENC10 are valid only for the ACR8010 and ACR8020 boards.
2. DACSTEP2 is currently valid only for the ACR1200 controller. This *configlist* argument defines an on-board single channel DAC output and an on-board single channel Stepper output hardware configuration.

3. DACSTEP4 is currently valid only for the ACR1500 controller. This *configlist* argument defines a hardware configuration of two channels of on-board DAC outputs and two channels of on-board Stepper outputs.

   The default hardware configuration for the ACR1200, ACR2000, ACR8000, ACR8010 boards is:

   ```
   CONFIG ENC8 DAC4 DAC4 ADC8****
   ```

   The default configuration for ACR1500 controller is:

   ```
   CONFIG ENC8 DAC4 DAC4 ADC8****
   CONFIG IO MODE 0
   CONFIG IO INPUT NEG
   CONFIG IO OUT NEG
   ```

   The default configuration for the ACR8020 controller is:

   ```
   CONFIG ENC8 DAC4 DAC4 ADC8****
   CONFIG XAXIS NONE NONE NONE NONE
   ```

   The default configuration for the ACR1505 controller is:

   ```
   CONFIG ENC8 DAC4 DAC4 ADC8****
   CONFIG XAXIS NONE NONE NONE NONE
   CONFIG IO MODE0
   CONFIG IO INPUT NEG
   CONFIG IO OUT NEG
   ```

4. The default hardware configuration is the same for all ACR Motion Control Boards. Since the hardware configuration of the ACR boards is user dependant, it is the user’s responsibility to set the correct hardware configuration.

Example 1 (ACR8000, ACR8010)

The following example defines six encoder channels on an ACR8000 or ACR8010 controller—a two-channel DAC module, a four-channel stepper module, and no analog input module:

```
CONFIG ENC6 DAC2 STEPPER4 NONE
```

Example 2 (ACR2000)

The following example defines four encoder channels, a two channel DAC module, and no analog input option on an ACR2000 controller:

NOTE: Module 1 is not used on an ACR2000 controller and should be set to “NONE”.

```
CONFIG ENC4 DAC2 NONE NONE
```

Example 3 (ACR1500)

The following example defines four encoder channels, four channels of on-board DAC outputs, and a 12-bit or 16-bit analog input module on an ACR1500 controller:

NOTE: Module 1 is not used on an ACR1500 controller and should be set to “NONE”.

```
CONFIG ENC4 DAC4 NONE ADC8
```

Example 4 (ACR1200)

The following example defines three encoder channels, an on-board single channel DAC output, an on-board single channel stepper output, and no analog input module on an ACR1200 controller:

NOTE: Module 1 is not used on an ACR1200 controller and should be set to "NONE".

```
CONFIG ENC3 DACSTEP2 NONE NONE
```

Example 5 (ACR8020)

The following example defines eight encoder channels, two four-channel DAC modules and eight ADC channels on an ACR8020 controller. The expansion board is configured to four encoder channels, four DAC channels, four stepper channels, and eight ADC channels.

```
CONFIG ENC8 DAC4 DAC4 ADC8
CONFIG XAXIS ENC4 DAC4 STEPPER4 ADC4
```

Example 6 (ACR1505)

The following example defines four encoder channels and four DAC outputs on an ACR1505 controller. The expansion board is configured to four encoder channels, four DAC channels, four stepper channels, and no ADC channels.

```
CONFIG ENC4 DAC4 NONE NONE
CONFIG XAXIS ENC4 DAC4 STEPPER4 NONE
```

CONFIG CLEAR Reset Default Configuration

Format	CONFIG CLEAR
Group	Operating System
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	ATTACH
Related Topics	N/A
Product Revision	Command and Firmware Release

NOTE: This command does not apply to the ACR9000 controller.

This command sets the hardware configuration to default.

The default configuration for ACR1200, ACR2000, ACR8000, ACR8010 boards is:

```
CONFIG ENC8 DAC4 DAC4 ADC8
```

The default configuration for ACR1500 controller is:

```
CONFIG ENC8 DAC4 DAC4 ADC8
CONFIG IO MODE 0
CONFIG IO INPUT NEG
CONFIG IO OUT NEG
```

The default configuration for the ACR8020 controller is:

```
CONFIG ENC8 DAC4 DAC4 ADC8****
CONFIG XAXIS NONE NONE NONE NONE
```

The default configuration for the ACR1505 controller is:

```
CONFIG ENC8 DAC4 DAC4 ADC8****
CONFIG XAXIS NONE NONE NONE NONE
CONFIG IO MODE0
CONFIG IO INPUT NEG
CONFIG IO OUT NEG
```

Example

The following example sets the hardware configuration to default and stores this information to EEPROM/FLASH:

```
CONFIG CLEAR
```

CONFIG IO Onboard IO Redirection

Format	CONFIG IO *input_destination output_source*
Group	Operating System
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	ATTACH
Related Topics	N/A
Product Revision	Command and Firmware Release

NOTE: You must use this command in conjunction with a **CONFIG XIO** command. See usage example below.

This command redirects the onboard digital IO. The *input_destination* argument tells the control where to place the bits read from the onboard inputs. The *output_source* argument tells the control where to get the bits that will be sent to the onboard outputs.

The default onboard IO redirection is:

```
CONFIG IO P4096 P4097
```

Example

The following example redirects onboard IO to Expansion Board 0 and Expansion Board 0 to onboard IO:

```
CONFIG IO P4104 P4105
CONFIG XIO0 P4096 P4097
```

CONFIG IO INPUT Configure Inputs Logic Polarity (ACR1500, ACR1505)

Format	CONFIG IO INPUT {*polarity*}
Group	Operating System
Units	N/A
Data Type	N/A
Default	Neg
Prompt Level	SYS, PROGx, PLCx
See Also	ATTACH
Related Topics	N/A
Product Revision	Command and Firmware Release

NOTE: This command applies only to the ACR1500 and ACR1505 controllers.

This command selects the input TTL logic polarity to be decoded by the controller. The polarity applies to all inputs. The *polarity* argument indicates the selected polarity as follows:

Polarity Argument	ON Logic Level	OFF Logic Level	Active Level
NEG	Logic Level Low	Logic Level High	Active Low
POS	Logic Level High	Logic Level Low	Active High

The default IO inputs polarity is:

```
CONFIG IO INPUT NEG
```

Example

The following example selects the IO inputs to positive logic polarity:

```
CONFIG IO INPUT POS
```

CONFIG IO MODE Onboard 82C55 IO Mode (ACR1500, ACR1505)

Format	CONFIG IO MODE {*io_mode*}
Group	Operating System
Units	N/A
Data Type	N/A
Default	0
Prompt Level	SYS, PROGx, PLCx
See Also	ATTACH
Related Topics	N/A
Product Revision	Command and Firmware Release

NOTE: This command applies only to the ACR1500 and ACR1505 controllers.

This command selects the Programmable Peripheral Interface IC (82C55) I/O mode of operation. The *io_mode* argument tells the control how to configure the 82C55 IC's input/output ports.

The factory default mode of operation is Mode 0 (24 Inputs, 24 Outputs).

The Bit Flag locations for the TTL compatible Digital I/O are mapped to the standard Input Bit Flags 0 through 31 (parameter P4096) and Output Bit Flags 32 through 63 (parameter P4097).

When the number of inputs or outputs configured exceeds 32 (ACR1500 only), they are mapped to the Expansion Input Bit Flags 256 through 271 (parameter P4104) and/or Expansion Output Bit Flags 288 through 303 (parameter P4105).

Controller	Modes
ACR1500	0-6
ACR1505	0-2

The following table provides the IO Mode configuration information, as well as the Bit Flag location for each group of inputs and outputs.

Config I/O Mode	I/O 00-07	I/O 08-15	I/O 16-23	I/O 24-31	I/O 32-39	I/O 40-47
0	Inputs Bit0-7	Inputs Bit8-15	Inputs Bit16-23	Outputs Bit32-39	Outputs Bit40-47	Outputs Bit48-55
1	Inputs Bit0-7	Inputs Bit8-15	Inputs Bit16-23	Outputs Bit32-39	Outputs Bit40-47	Inputs Bit24-31
2	Inputs Bit0-7	Inputs Bit8-15	Outputs Bit56-63	Outputs Bit32-39	Outputs Bit40-47	Outputs Bit48-55
3	Inputs Bit0-7	Inputs Bit8-15	Inputs Bit16-23	Outputs Bit32-39	Inputs Bit256-263	Inputs Bit24-31

Config I/O Mode	I/O 00-07	I/O 08-15	I/O 16-23	I/O 24-31	I/O 32-39	I/O 40-47
4	Inputs Bit0-7	Outputs Bit288-295	Outputs Bit56-63	Outputs Bit32-39	Outputs Bit40-47	Outputs Bit48-55
5	Inputs Bit0-7	Inputs Bit8-15	Inputs Bit16-23	Inputs Bit264-271	Inputs Bit256-263	Inputs Bits24-31
6	Outputs Bit296-303	Outputs Bit288-295	Outputs Bit56-63	Outputs Bit32-39	Outputs Bit40-47	Outputs Bit48-55

ACR1500: Uses modes 0-6.

ACR1505: Uses modes 0-3.

Example

The following example sets the IO Mode to Mode 2 (16 Inputs / 32 Outputs):

```
CONFIG IO MODE 2
```

CONFIG IO OUT Configures Outputs Logic Polarity (ACR1500, ACR1505)

Format	CONFIG IO OUT {*polarity*}
Group	Operating System
Units	N/A
Data Type	N/A
Default	Neg
Prompt Level	SYS, PROGx, PLCx
See Also	ATTACH
Related Topics	N/A
Product Revision	Command and Firmware Release

NOTE: This command applies only to the ACR1500 and ACR1505 controllers.

This command selects the output TTL logic polarity to be decoded by the controller. The polarity applies to all outputs. The *polarity* argument indicates the selected polarity as follows:

Polarity Argument	ON Logic Level	OFF Logic Level	Active Level
NEG	Logic Level Low	Logic Level High	Active Low
POS	Logic Level High	Logic Level Low	Active High

The default IO outputs polarity is:

```
    CONFIG IO OUT NEG
```

Example

The following example selects the IO outputs to positive logic polarity:

```
CONFIG IO OUT POS
```

CONFIG XIO Expansion IO Redirection

Format	CONFIG XIO board *input_destination output_source*
Group	Operating System
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	ATTACH
Related Topics	N/A
Product Revision	Command and Firmware Release

NOTE: You must use this command in conjunction with a **CONFIG IO** command. See usage example below.

This command redirects the expansion digital IO. The *board* argument indicates which expansion IO board is to be redirected. The *input_destination* argument tells the control where to place the bits read from the expansion inputs. The *output_source* argument tells the control where to get the bits that will be sent to the expansion outputs.

The default expansion IO redirections are:

```
CONFIG XIO0 P4104 P4105
CONFIG XIO1 P4106 P4107
CONFIG XIO2 P4108 P4109
CONFIG XIO3 P4110 P4111
```

Example

The following example redirects onboard IO to Expansion Board 0 and Expansion Board 0 to onboard IO:

```
CONFIG IO P4104 P4105
CONFIG XIO0 P4096 P4097
```

CPU Display Processor Loading

Format	CPU
Group	Operating System
Units	Percent
Data Type	Long
Default	N/A
Prompt Level	N/A
See Also	PERIOD, DIAG
Related Topics	N/A
Product Revision	Command and Firmware Release

This command displays the processor load as a percentage of the foreground and background timing. Background time consists of servo loop updates, velocity profiles and axis position interpolation. Foreground time is the time left over for the execution of user programs. High foreground percentages usually mean that the user programs are going to execute faster.

This command is used along with the **PERIOD** command to control the foreground/background percentages in the system.

Example

```
CPU
```

DAC Analog Output Control

Format 1:	DAC *index* GAIN {*gain*}
Format 2:	DAC *index* OFFSET {*offset*}
Group	Global Objects
Units	See below
Data Type	FP32
Default	Gain 3276.8 counts/volt Offset 0.0 Volts
Prompt Level	N/A
See Also	AXIS, ENC
Related Topics	N/A
Product Revision	Command and Firmware Release

The **DAC** commands give direct access to the D/A converter software adjustments.

Issuing these commands without the final argument will display their current settings.

The default **GAIN** is 3276.8 DAC units/volt and the default **OFFSET** is 0.0 volts.

Note that the output voltage is inverted in the output stage of the hardware, therefore the default **DAC GAIN** will physically send out a negative voltage for positive settings.

Example

The following example sets offset on **DAC3** to 125 millivolts:

```
DAC3 OFFSET 0.125
```

DEC Set Deceleration Ramp

Format	DEC {*rate*}
Group	Velocity Profile
Units	units/second^2 based on PPU
Data Type	FP32
Default	20000
Prompt Level	PROGx
Report Back	Yes **To view, type DEC**
See Also	ACC, FVEL, IVEL, MASTER, STP, VEL, PPU
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Sets the master deceleration for ramping from higher to lower speeds for coordinated motion.

Arguments

rate

Optional. Sets the deceleration rate for the master profile.

Remarks

Use **DEC** to blend back-to-back moves (where **STP0**). The coordinated motion profiler uses the **DEC** ramp to merge from a higher to lower commanded velocity at the start of the next move.

NOTE: Though you might not directly use **DEC** in your application, it is a good idea to set a **DEC** rate. A variety of bits use the **DEC** rate when pausing or stopping motion. Consider a reasonable rate, perhaps the same rate as **STP** when used to stop motion or blend into **FVEL**.

Disabling Deceleration Ramp

With the *rate* set to zero, you can completely disable the deceleration ramp. If the motion needs to slow down (for example, sending **FOV**, thereby changing the motion profile velocity), the motion profiler uses infinite deceleration, which is nearly instantaneous.

► To disable the deceleration ramp, set the *rate* to zero.

DEC versus STP

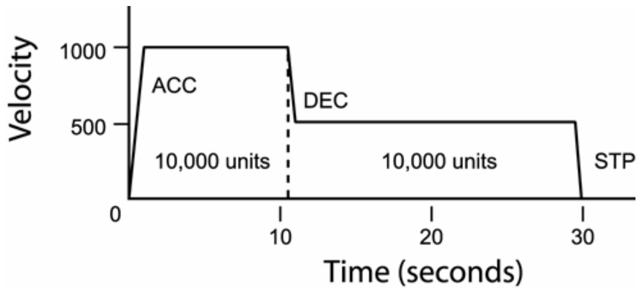
The table below summarizes the circumstances where **DEC** or **STP** are used. The **DEC** command can affect **CIRCCW**, **CIRCW**, **INT**, **MOV**, **PAUSE**, **SINE**, **STP**, and **TRJ**.

Description	DEC	STP
A Master's "Stop all Moves Request" bit is set. For example, Bit523 for Master 0.	Yes	No
A Master's "Feedhold Request" bit is set. For example, Bit520 for Master 0.	Yes	No
A program issues a **PAUSE**, which generates a "Feedhold Request" for the attached master.	Yes	No
A Master's **STP** rate is zero, and the next move segment's `VEL` rate is lower than the current segment's.	Yes	No
A Master's **FOV** rate is changed to a lower value during a move segment.	Yes	No
A Master's **ROV** rate is changed to a lower value during a move segment, and the "Rapid Active" bit is set.	Yes	No
When using **TRJ** moves.	Yes	No
Interpolated motion profile completes.	No	Yes

NOTE: For the ACR9000, hardware and software limits use the **HLDEC** and **SLDEC** for deceleration on contacting an end-of-travel limit. These rates override the **DEC** and **STP** values.

Example

The following example sets up a simple motion profile. The first move accelerates to velocity and moves 10,000 incremental units. At the start of the second move, the motor ramps down to the new velocity using the previously stated deceleration rate. At the end of the second move, the stop ramp ends motion.



```
DEC 1000 ACC 1000 STP0 VEL1000
X/10000   : REM move 10,000 incremental units
STP1000 VEL500
X/10000   : REM move 10,000 incremental units
```

DEF Display the Defined Variable

Format	DEF {*number*}
Group	Operating System
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	#DEFINE, DIM
Related Topics	N/A
Product Revision	Command and Firmware Release

The **DEF** command displays the currently defined user variables. You must set **DIM DEF** to a nonzero value to define user variables.

Example

```
SYS>DEF
#DEFINE LED BIT96
#DEFINE MYFLAG BIT32
#DEFINE TRUE 1
#DEFINE COUNTER LV2
#DEFINE LOOP LV4
```

#DEFINE Define Aliases

Format	#DEFINE {*name*} {*parameter*}
Group	Operating System
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	DEF, DIM
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

The **DEFINE** command lets you assign user names to parameters, bits, constants, and variables. Aliases are recognized globally across user programs. Aliases can be used to make program code more readable.

NOTE: You must set **DIM DEF** to a nonzero value to define user variables. Use ACR-View's configuration wizard to set the number of desired defines.

Remarks

Changing a Define name or assigned variable requires the **CLEAR** command at the system level. **CLEAR** will erase **PROGRAM** and **DEF** dimensioning. The easiest way to accomplish this is to download the Configuration, which includes **CLEAR**, **DIM PROG**, and **DIM DEF** statements. Downloading configuration will erase all programs from the controller.

#DEFINE statements should be placed in the ACR-View (release 5.5 or later) Defines editor, or before the **PROGRAM** statement in any program editor. **#DEFINE** statements should not be placed with a Program.

Observe the following rules when creating and using aliases:

- You can use a maximum of 24 letters.
- Aliases are case sensitive.
- Do not use numbers, spaces, or special characters (such as _ and @).
- Constants can only be positive integers. Negative numbers and floating point values will be truncated.
- Use care when assigning aliases to local variables:

  An alias is recognized across all programs, while local variables are limited to the program in which they are created. This can cause problems if you have created similar local variables in different programs. For example, suppose you create long variables in three programs, and then assign the alias "counter" to LV1 (long variable 1). The controller recognizes "counter" as an alias in all three programs, though it represents a counter in only one program.

- Do not use reserved keywords. Type `HELP` in the terminal emulator to see a list of reserved keywords for the ACR controller.

Examples

Example 1

```
#DEFINE bOutGripper BIT32
#DEFINE pSysClock P6916
#DEFINE pActualPositionX P12290
#DEFINE bHomeInProgressX BIT128
#DEFINE bHomeCompleteX BIT16134
#DEFINE bHomeFailedX BIT16135
#DEFINE lvCounter LV0
#DEFINE cValidConstant 10
#DEFINE cInvalidConstant 1.1
#DEFINE cINVALIDCONSTANT -1

PROGRAM
PRINT "pSysClock",pSysClock
PRINT "psysclock",psysclock
PRINT "cValidConstant", cValidConstant
PRINT "cInvalidConstant",cInvalidConstant
PRINT "cINVALIDCONSTANT",cINVALIDCONSTANT
ENDP
```

Downloading this program and define statement will result in an Unknown command in line 30 error because the *psysclock* is not the same case as the *pSysClock*. Listing the program will look like this:

```
P00>list
 PROGRAM
 PRINT "pSysClock",pSysClock
 PRINT "cValidConstant",10
 PRINT "cInvalidConstant",1
 PRINT "cINVALIDCONSTANT",0
 ENDP
```

LRUN the program to view the values:

```
P00>LRUN
pSysClock      123101747
cValidConstant 10
cInvalidConstant       1
cINVALIDCONSTANT       0
P00>
```

Notice that *cInvalidConstant* is reported as 1, rather than 1.1 and *cINVALIDCONSTANT* is 0 rather than –1. Floats and negative numbers can be assigned to variables. See edited program below with new DEFINEs added.

```
#DEFINE NegativeOne     P1
#DEFINE Float   P2

P00>LIST
 PROGRAM
 PRINT "pSysClock",pSysClock
 PRINT "cValidConstant",10
 NegativeOne= -1
 Float= 1.1
 PRINT "NegativeOne",NegativeOne
 PRINT "Float",Float
 ENDP
P00>LRUN
pSysClock      123810189
cValidConstant 10
NegativeOne    -1
Float   1.1
P00>
```

DETACH Clear Attachments

Format	DETACH {ALL}
Group	Operating System
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	ATTACH
Related Topics	N/A
Product Revision	Command and Firmware Release

This command cancels the master and slave attachments created with the **ATTACH** command. The **ATTACH** and **DETACH** commands can be issued from within a program, but special care must be taken to prevent errors that will halt the program.

The **DETACH ALL** command can be issued from anywhere in the system and will detach all slaves from all masters and all masters from all programs.

Example

The following example detaches the master and slaves from the current program:

```
DETACH
```

DGAIN Set Derivative Gain

Format	DGAIN {*axis* {*value*}} {*axis* {*value*}} …
Group	Servo Control
Units	volts / pulses / second
Data Type	FP32
Default	0
Prompt Level	PROGx
See Also	DWIDTH, FFVEL, FFACC, IGAIN, PGAIN
Related Topics	Axis Parameters (0-7) (8-15)
Product Revision	Command and Firmware Release

This command modifies the value used in the PID algorithm to control derivative gain. Issuing a **DGAIN** command to an axis without an argument will display the current setting for that axis. The default gain is 0.0 for all axes.

Example

The following example sets X axis derivative gain to 0.0001 volts/pulses/second:

```
DGAIN X0.0001
```

DGAIN Smooth

Take away humming noise from the torque motor due to **DGAIN**.

Axis parameter "DGAIN Smooth" is used to subdue the humming noise in the torque loop due to **DGAIN**. The Default value is 0, which means no smoothing is applied. The user may typically change this value from 0 to 5. The **DGAIN** command must be used after changing this parameter to make this change effective.

Example

```
REM The dgain term will be averaged over 4 samples.
P12402 = 4
DGAIN X0.0001
```

DIAG Display System Diagnostics

Format	DIAG
Group	Operating System
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	CPU, PERIOD
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Displays the status of various power conditions, as well as indicates the option modules present for the ACR1200, ACR1500, ACR1505, ACR2000, ACR8000, ACR8010, ACR8020, and ACR9000 controllers.

Remarks

In Firmware Version 1.18 and up, these status bits are also available as a parameter, P7044 (See the *ACR Parameter and Bit Reference*.)

If any tests fail, check all fuses and external voltage inputs on the controller (as the first step in troubleshooting the problem).

The following describes the command results for the ACR controllers supporting the **DIAG** command, including the ACR1200, ACR2000 and the ACR8010 option modules.

ACR1200 Controller DIAG Command Definitions

The following will be displayed when invoking the **DIAG** command on an ACR1200 Controller.

```
EXT: PASS
ISO: PASS
VDD: PASS
VEE: PASS
BCL: PASS
BCF: PASS
ENC: PASS
STP: PASS
```

EXT: Isolated external voltage supplied for the opto-isolation circuitry on the ACR1200 Motherboard.

PASS: External voltage present

FAIL: Insufficient external voltage

ISO: On-board isolated +5 VDC voltage provided for the opto-isolation circuitry on the ACR1200 Motherboard. The isolated +5 VDC is generated from the isolated, externally supplied voltage—it is not the user-supplied +5V.

PASS: On-board isolated +5 VDC voltage present

FAIL: Insufficient voltage

VDD:	+12 VDC supply voltage.
	PASS: Voltage present
	FAIL: Insufficient voltage
VEE:	-12 VDC supply voltage.
	PASS: Voltage present
	FAIL: Insufficient voltage
BCL:	1000 mAh Lithium Battery BT1 voltage low indicator. This is a warning indicator that battery voltage is approaching minimum requirements for SRAM back-up. Minimum SRAM data retention voltage is 2.0 VDC. BT1 should be replaced. (Panasonic P/N CR2477N)
	PASS: BT1 > 2.5 VDC
	FAIL: BT1 is between 2.3 and 2.5 VDC (when BCF displays PASS)
BCF:	1000 mAh Lithium Battery BT1 voltage fail indicator. This is a warning indicator that battery voltage is below requirements for SRAM back-up (minimum SRAM data retention voltage is 2.0 VDC). BT1 must be replaced. (Panasonic P/N CR2477N)
	PASS: BT1 > 2.2VDC
	FAIL: BT1 is between 2.0 and 2.2 VDC.
ENC:	Fused Encoder +5 VDC available from the P1 encoder connector.
	PASS: Voltage present
	FAIL: Insufficient voltage
STP:	Fused Stepper +5 VDC available from the P2 analog connector.
	PASS: Voltage present
	FAIL: Insufficient voltage

ACR1500 Controller DIAG Command Definitions

The following will be displayed when invoking the **DIAG** command on an ACR1500 Controller.

Encoder Power

```
EVCC: PASS
```

EVCC:	Fused +5 VDC available from the P1 Encoder connector.	
	PASS:	Voltage present
	FAIL:	Insufficient voltage

ACR1505 Controller DIAG Command Definitions

The following will be displayed when invoking the **DIAG** command on an ACR1505 Controller.

```
Opto-isolated Power
ENC_PWR: PASS
STP_PWR: PASS
IO_PWR:  PASS
```

ENC_PWR:	Fused +5 VDC available from the P1 Encoder connector.	
	PASS:	Voltage present
	FAIL:	Insufficient voltage
STP_PWR:	Fused +5 VDC available from the P2 Analog I/O connector.	
	PASS:	Voltage present
	FAIL:	Insufficient voltage
IO_PWR:	Fused +5 VDC available from the P3/P4 I/O connectors.	
	PASS:	Voltage present
	FAIL:	Insufficient voltage

ACR2000 Motherboard DIAG Command Definitions

The following will be displayed when invoking the **DIAG** command on an ACR2000 Controller.

Opto-isolated Power

```
EXT: PASS
ISO: PASS
```

EXT:	+24V isolated external voltage supplied for the opto-isolation circuitry on the ACR2000 Motherboard.	
	PASS:	External voltage present
	FAIL:	Insufficient external voltage
ISO:	On-board isolated +5 VDC voltage provided for the opto-isolation circuitry on the ACR2000 Motherboard. The isolated +5 VDC is generated from the isolated, externally supplied voltage. **NOTE:** It is not the +5V supplied by the user.	
	PASS:	On-board isolated +5 VDC voltage present
	FAIL:	Insufficient voltage

ACR2000 ACRCOMM Com Board DIAG Command

The following will be displayed when invoking the **DIAG** command on an ACR2000 Controller with an ACRCOMM module.

COMM Board Detected

```
BID: 0
VDD: PASS
VEE: PASS
BCL: PASS
BDF: PASS
```

BID:	Board ID number for a COMM board.	
	0: COMM Board ID Number	
VDD:	+12 VDC supply voltage.	
	PASS:	Voltage present
	FAIL:	Insufficient voltage
VEE:	-12 VDC supply voltage.	
	PASS:	Voltage present
	FAIL:	Insufficient voltage
BCL:	1000 mAh Lithium Battery BT1 voltage low indicator. This is a warning indicator that battery voltage is approaching minimum requirements for SRAM back-up. Minimum SRAM data retention voltage is 2.0VDC. BT1 should be replaced. (Panasonic P/N CR2477N)	
	PASS:	BT1 > 2.5 VDC
	FAIL:	BT1 is between 2.3 and 2.5 VDC (when BCF displays PASS)
BCF:	1000 mAh Lithium Battery BT1 voltage fail indicator. This is a warning indicator that battery voltage is below requirements for SRAM back-up (minimum SRAM data retention voltage is 2.0VDC). BT1 must be replaced. (Panasonic P/N CR2477N)	
	PASS:	BT1 > 2.2VDC
	FAIL:	BT1 is between 2.0 and 2.2 VDC.

ACR8000 Controller DIAG Command Definitions

The following will be displayed when invoking the **DIAG** command on an ACR8000 Controller.

```
+24V: PASS
+5V: PASS
+12V: PASS
-12V: PASS
```

+24V:	Isolated external voltage supplied for the opto-isolation circuitry on the ACR8000 controller.
	PASS: External voltage present
	FAIL: Insufficient external voltage
+5V:	On-board isolated +5 VDC voltage provided for the opto-isolation circuitry on the ACR8000 controller. The isolated +5 VDC is generated from the isolated, externally supplied voltage—it is not the +5V supplied by the user.
	PASS: On-board isolated +5 VDC voltage present
	FAIL: Insufficient voltage
+12V:	+12 VDC supply voltage.
	PASS: Voltage present
	FAIL: Insufficient voltage
-12V:	-12 VDC supply voltage.
	PASS: Voltage present
	FAIL: Insufficient voltage

ACR8010 Motherboard DIAG Command Definitions

The following will be displayed when invoking the **DIAG** command on an ACR8010 Controller.

Opto-isolated Power

```
EXT: PASS
ISO: PASS
VDD: PASS
VEE: PASS
BCL: PASS
BCF: PASS
```

EXT:	Isolated external voltage provided for the opto-isolation circuitry on the ACR8010 Motherboard.
	PASS: External voltage present
	FAIL: Insufficient external voltage
ISO:	On-board isolated +5 VDC voltage provided for the opto-isolation circuitry on the ACR8010 Motherboard. The isolated +5 VDC is generated from the isolated, externally supplied voltage—it is not the +5V supplied by the user.
	PASS: On-board isolated +5 VDC voltage present
	FAIL: Insufficient voltage
VDD:	+12 VDC supply voltage.
	PASS: Voltage present
	FAIL: Insufficient voltage
VEE:	-12 VDC supply voltage.
	PASS: Voltage present
	FAIL: Insufficient voltage

BCL:	1000 mAh Lithium Battery BT1 voltage low indicator. This is a warning indicator that battery voltage is approaching minimum requirements for SRAM back-up. Minimum SRAM data retention voltage is 2.0VDC. BT1 should be replaced. (Panasonic P/N CR2477N)
	PASS: BT1 > 2.5 VDC
	FAIL: BT1 is between 2.3 and 2.5 VDC (when BCF displays PASS)
BCF:	1000 mAh Lithium Battery BT1 voltage fail indicator. This is a warning indicator that battery voltage is below requirements for SRAM back-up (minimum SRAM data retention voltage is 2.0VDC). BT1 must be replaced. (Panasonic P/N CR2477N)
	PASS: BT1 > 2.2VDC
	FAIL: BT1 is between 2.0 and 2.2 VDC.

ACR8020 Controller DIAG Command Definitions

The following will be displayed when invoking the DIAG command on an ACR8020 Controller.

Opto-isolated Power

```
EXT:  PASS
ISO:  PASS
VDD:  PASS
VEE:  PASS
```

EXT:	+24V isolated external voltage supplied for the opto-isolation circuitry on the ACR8020 Motherboard.
	PASS: External +24V present
	FAIL: Insufficient external voltage
ISO:	On-board isolated +5 VDC voltage supplied for the opto-isolation circuitry on the ACR8020 Motherboard. The isolated +5 VDC is generated from the isolated, externally supplied voltage—it is not the +5V supplied by the user.
	PASS: On-board isolated +5 VDC voltage present
	FAIL: Insufficient voltage
VDD:	+12 VDC supply voltage.
	PASS: Voltage present
	FAIL: Insufficient voltage
VEE:	-12 VDC supply voltage.
	PASS: Voltage present
	FAIL: Insufficient voltage

ACR9000 Controller DIAG Command Definitions

When using an ACR9000, the **DIAG** command reports the status for Battery Backup for RAM, CANopen I/O, and IP connection client command streams, IP connection fast status, and IP connection client management (watchdogs). You can also use the following commands to tailor the information you want tot view:

For the ACR9000, the following are valid command to tailor the information you want to view:

DIAG XIO: View CANopen I/O status.

DIAG IP: View connectivity status (not including CANopen).

DIAG IP STREAM: View only command stream IP connectivity status.

DIAG IP FSTAT: View only fast status IP connectivity status.

DIAG IP EXC: View only management IP connectivity status.

NOTE: The **DIAG** command does not provide information about the ACR9000 controller voltages.

Example

The following example is displayed for an ACR9000 controller with Analog Inputs, CANopen I/O, and Ethernet.

```
BCL: PASS
ANI Card installed
CAN Controller installed, user modifiable parameters in parentheses, e.g.
(P32768)
Network reset or not started
Master Node ID (P32768) = 5
Bit Rate in KBaud (P32769) = 125
Number of external nodes (P32770) = 1
Cyclic Period (milliseconds) (P32772) = 50
Health Period (milliseconds) = 500
Total Digital input bytes = 0
Total Digital output bytes = 0
Total Analog input points = 0
Total Analog output points = 0
CANopen bus state = 1 (pre-operational, ready to start)
Data for external no
     Node ID (P33024) = 1
     Digital input bytes = 0
     Digital output bytes = 0
     Analog input points = 0
     Analog output points = 0
     Health type = 0 (lifeguarding)
     Node state = 0 (not responding)

Ethernet Controller installed

Data for Client Command Connection #  (0-4)

Data for Client Command Connection #0
Connection status = 2 (currently connected, watchdog protected)
Client's IP address = "172.20.129.187"
Client's IP port = 1234
current status duration = 755 (00:12:35)
current/recent attached ACR stream # = 1
connection guarded by watchdog # 1
connection not guarded by watchdog
```

```
Data for Client Command Connection #1
Connection status = 0 (never connected)

Data for Client Command Connection #2
Connection status = 0 (never connected)

Data for Client Command Connection #3
Connection status = 0 (never connected)

Data for Client Command Connection #4
Connection status = 0 (never connected)

Data for Fast Status Sender #0

Subscription status = 2 (currently subscribed to, watchdog protected)

Client's IP address = 172.20.129.187
Client's IP port = 1234
current status duration = 755 (00:12:35)
Update period to client = 100
Subscription guarded by watchdog # 1
Subscription not guarded by watchdog

Data for Fast Status Sender #1
Subscription status = 0 (never subscribed to)

Data for Fast Status Sender #2
Subscription status = 0 (never subscribed to)

Data for Fast Status Sender #3
Subscription status = 0 (never subscribed to)

Data for Fast Status Sender #4
Subscription status = 0 (never subscribed to)

Data for Client Management Connection # (0-4)

Connection status = 1 (currently connected, no watchdog)

Data for Client Management Connection #0
Client's IP address = 172.20.129.187
Client's IP port = 1234
current status duration = 755 (00:12:35)
watchdog timer value = 200
watchdog ticker value = 3
user alarm filter value = 7
controller alarm filter value = 12

Data for Client Management Connection #1
Connection status = 0 (never connected)

Data for Client Management Connection #2
Connection status = 0 (never connected)

Data for Client Management Connection #3
Connection status = 0 (never connected)

Data for Client Management Connection #4
Connection status = 0 (never connected)
```

BCL: 210000 mAh Lithium Battery voltage low indicator. This is a warning indicator that battery voltage is approaching minimum requirements for SRAM back-up. Minimum SRAM data retention voltage is 2.0 VDC. Factory service is required for battery replacement. (Tadiran P/N TL-5242/W)

PASS: Always for an old board, or if there is a good battery in a board that supports battery backed RAM.

FAIL: If there is no battery or the battery is low in a board that supports battery backed RAM. The fail will latch on power-up if the battery <2.2 VDC.

ACRIO Expansion I/O Board DIAG Command Definitions

If an ACRIO Expansion I/O board is present (ACR1200, ACR2000, ACR8010, ACR8020, and ACR9000) the following diagnostics are displayed. This is in addition to the diagnostics displayed for the controller.

When multiple expansion I/O boards are present, the board number is listed with diagnostic information present for each board.

```
XIO Board "#" Detected
BID: 16
EXT: PASS
ISO: PASS
```

#:	XIO Board Number (1 through 4) as selected by J1 and J2 on the ACRIO boards.	
BID:	Board ID number for an ACRIO board. This number is the same for all XIO boards.	
	16: ACRIO Board ID Number	
EXT:	Isolated external voltage provided for the opto-isolation circuitry on the ACRIO board.	
	PASS:	External voltage present
	FAIL:	Insufficient external voltage
ISO:	ISO: On-board isolated +5 VDC voltage provided for the opto-isolation circuitry on the ACRIO board. The isolated +5 VDC is generated from the isolated, externally supplied voltage—it is not the +5V supplied by the user.	
	PASS:	On-board isolated +5 VDC voltage present
	FAIL:	Insufficient voltage

Example

```
DIAG
```

DIAG EPLD Display ETHERNET Powerlink Diagnostics

Format	DIAG EPLD
Group	Operating System
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	EPLC, OPEN_EPLD
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Reports the current values of the ETHERNET Powerlink (EPL) network, and node status bits and parameters.

Remarks

This command also provides some brief descriptions, and includes all setup values and current status.

Example

The following example is displayed for an ACR9030 with three EPL drives.

```
SYS>diag epld
EPL Controller installed, user modifiable parameters in parentheses, e.g.
(P37376)
Network operational
Number of Controlled nodes (P37376) = 3
EPL network state = 5 (EPL operational)
current operational duration = 4 (0:0:4)
Data for external node 0
   Node ID (P37632) = 10
   Drive Mode (P37634) = 7
   TPC/IP Port (P37635) = 5002
   Attached Axis # = 0
   Node status = 1 (Node found)
      failure count = 0, operational duration = 4 (0:0:4)
Data for external node 1
   Node ID (P37648) = 11
   Drive Mode (P37650) = 7
   TPC/IP Port (P37651) = 5002
   Attached Axis # = 1
   Node status = 1 (Node found)
      failure count = 0, operational duration = 4 (0:0:4)
Data for external node 2
   Node ID (P37664) = 12
   Drive Mode (P37666) = 7
   TPC/IP Port (P37667) = 5002
   Attached Axis # = 2
   Node status = 1 (Node found)
      failure count = 0, operational duration = 5 (0:0:5)
SYS>
```

DIAG IP Display System Diagnostics

Format	DIAG IP
Group	Operating System
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	CPU, PERIOD
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Displays the IP connectivity status.

Remarks

Display does not include CANopen I/O status.

Typing **DIAG IP** restricts the **DIAG** report to only display IP connectivity status. This includes all Client Command, Fast Status, and Client Management connections.

The same info can be displayed by typing **DIAG IP STREAM**, **DIAG IP FSTAT**, and **DIAG IP EXC** in succession.

Example

```
P00>DIAG IP
Ethernet Controller MAC address 00 90 55 FF EF 2D

Data for Client Command Connection #0
Connection status = 2 (currently connected, watchdog protected)
Client's IP address = "172.26.129.214"
Client's IP port = 1155
current status duration = 67 (0:1:7)
current/recent attached ACR stream # = 2
connection guarded by watchdog #0

Data for Client Command Connection #1
Connection status = 3 (connection closed by client)
Client's IP address = "172.26.129.214"
Client's IP port = 1440
current status duration = 602624 (167:23:44)
current/recent attached ACR stream # = 0
connection not guarded by watchdog

Data for Client Command Connection #2
Connection status = 0 (never connected)

Data for Client Command Connection #3
Connection status = 0 (never connected)

Data for Client Command Connection #4
Connection status = 0 (never connected)

Data for Fast Status Sender #0
Subscription status = 0 (never subscribed to)

Data for Fast Status Sender #1
Subscription status = 0 (never subscribed to)

Data for Fast Status Sender #2
```

```
Subscription status = 0 (never subscribed to)

Data for Fast Status Sender #3
Subscription status = 0 (never subscribed to)

Data for Fast Status Sender #4
Subscription status = 0 (never subscribed to)

Data for Client Management Connection #0
Connection status = 1 (currently connected)
Client's IP address = "172.26.129.214"
Client's IP port = 1156
current status duration = 1 (0:0:1)
watchdog timer value = 2000
watchdog ticker value = 4
user alarm filter value = 0
controller alarm filter value = 0

Data for Client Management Connection #1
Connection status = 3 (connection closed by client)
Client's IP address = "172.26.129.225"
Client's IP port = 1327
current status duration = 15281 (4:14:41)
watchdog timer value = 0
watchdog ticker value = 0
user alarm filter value = 0
controller alarm filter value = 0

Data for Client Management Connection #2
Connection status = 3 (connection closed by client)
Client's IP address = "172.26.129.225"
Client's IP port = 1528
current status duration = 602349 (167:19:9)
watchdog timer value = 0
watchdog ticker value = 0
user alarm filter value = 0
controller alarm filter value = 0

Data for Client Management Connection #3
Connection status = 0 (never connected)

Data for Client Management Connection #4
Connection status = 0 (never connected)
```

DIAG IP EXC Display IP Connectivity Status

Format	DIAG IP EXC
Group	Operating System
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	CPU, PERIOD
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Displays the management IP connectivity status.

Remarks

Typing **DIAG IP EXC** restricts the **DIAG** report to only display the status of the IP management connections.

Example

```
SYS> DIAG IP EXC
Ethernet Controller MAC address 00 90 55 00 73 83

Data for Client Management Connection #0
Connection status = 1 (currently connected)
Client's IP address = "192.168.10.10"
Client's IP port = 1234
current status duration = 1 (0:0:1)
watchdog timer value = 2000
watchdog ticker value = 4
user alarm filter value = 0
controller alarm filter value = 0

Data for Client Management Connection #1
Connection status = 0 (never connected)

Data for Client Management Connection #2
Connection status = 0 (never connected)

Data for Client Management Connection #3
Connection status = 0 (never connected)

Data for Client Management Connection #4
Connection status = 0 (never connected)
```

DIAG IP FSTAT Display System Diagnostics

Format	DIAG IP FSTAT
Group	Operating System
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	CPU, PERIOD
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Displays the fast status IP connectivity status.

Remarks

Typing **DIAG IP FSTAT** restricts the **DIAG** report to only display information that pertains to Fast Status connections.

Example

```
SYS> DIAG IP FSTAT
Ethernet Controller MAC address 00 91 55 00 74 83

Data for Fast Status Sender # 0

Subscription status = 2 (currently subscribed to, watchdog protected)

Client's IP address = 172.20.129.187
Client's IP port = 1234
current status duration = 755 (00:12:35)
Update period to client = 100
Subscription guarded by watchdog # 1
Subscription not guarded by watchdog

Data for Fast Status Sender #1
Subscription status = 0 (never subscribed to)

Data for Fast Status Sender #2
Subscription status = 0 (never subscribed to)

Data for Fast Status Sender #3
Subscription status = 0 (never subscribed to)

Data for Fast Status Sender #4
Subscription status = 0 (never subscribed to)
```

DIAG IP STREAM Display System Diagnostics

Format	DIAG IP STREAM
Group	Operating System
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	CPU, PERIOD
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Displays the IP stream connectivity status.

Remarks

Typing **DIAG IP STREAM** narrows the **DIAG** report to only display system diagnostics pertaining to IP stream connectivity.

Example

```
SYS>DIAG IP STREAM
Ethernet Controller MAC address 00 90 55 00 37 83

Data for Client Command Connection #0
Connection status = 2 (currently connected, watchdog protected)
Client's IP address = "192.168.10.10"
Client's IP port = 1547
current status duration = 8644 (2:24:4)
current/recent attached ACR stream # = 2
connection guarded by watchdog #0

Data for Client Command Connection #1
Connection status = 0 (never connected)

Data for Client Command Connection #2
Connection status = 0 (never connected)

Data for Client Command Connection #3
Connection status = 0 (never connected)

Data for Client Command Connection #4
Connection status = 0 (never connected)
```

DIAG XIO Display System Diagnostics

Format	DIAG XIO
Group	Operating System
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	CPU, PERIOD
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Displays the CANopen I/O status.

Remarks

Typing **DIAG XIO** narrows the **DIAG** report to only display the status of the external I/O (CANopen).

Example

```
SYS> DIAG XIO
CAN Controller installed, user modifiable parameters in parentheses, e.g.
(P32768)
Network reset or not started
Master Node ID (P32768) = 5
Bit Rate in KBaud (P32769) = 125
Number of external nodes (P32770) = 1
Digital I/O mapping option (P32771) = 0
Cyclic Period (milliseconds) (P32772) = 50
Health Period (milliseconds) (P32773) = 500
Total Digital input bytes = 0
Total Digital output bytes = 0
Total Analog input points = 0
Total Analog output points = 0
CANopen bus state = 1 (pre-operational, ready to start)
Data for external node 0
     Node ID (P33024) = 1
     Digital input bytes = 0
     Digital output bytes = 0
     Analog input points = 0
     Analog output points = 0
     Health type (P33029) = -1 (undetermined)
     Node state = 0 (not responding)
```

DIM Allocate Memory

Format	DIM
Group	Memory Control
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx
See Also	CLEAR, MEM
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Issuing the **DIM** command alone provides a report of the current memory allocations.

Remarks

This command is used along with a second command to allocate memory space for programs, buffers, variables, and arrays.

The following is a list of valid command combinations:

Issued at the System Prompt

DIM COM1	Allocate COM1 buffer.
DIM COM2	Allocate COM2 buffer.
DIM DEF	Allocate memory for defined variables. (See the #DEFINE command.)
DIM DPCB	Allocate DPCB buffer.
DIM FIFO	Allocate FIFO buffer.
DIM LOGGING	Allocate non-volatile, battery back-up memory for logging parameters.
DIM P	Allocate global variables (64-bit floating points).
DIM PLC	Allocate PLC memory.
DIM PROG	Allocate program memory.

Issued at the Program Prompt (Locals)

DIM $A	Allocate string array references and string arrays.
DIM DA	Allocate double-precision array references and double-precision arrays.
DIM DV	Allocate double-precision floating-point variables.
DIM LA	Allocate long array references and long arrays.
DIM LV	Allocate long variables.
DIM MBUF	Allocate move buffers for motion profiler.

DIM SA	Allocate single-precision array references and single-precision arrays.
DIM SV	Allocate single-precision floating-point variables.
DIM $V	Allocate strings.

The total RAM available for user allocation is as follows:

- 64k× 8 bytes for the ACR8000
- 128k× 8 bytes for the ACR1200, ACR1500, and ACR2000
- 512k× 8 bytes for the ACR1505, ACR8010, ACR8020 (1-8 axes), and expanded Memory ACR2000
- 1024k x 8 bytes for ACR9000
- 2048k x 8 bytes for ACR8020 (9-16 axes) and ACR8010 (firmware revision 1.18.12)

The following shows memory usage by various data and program structures:

Data/Program Structure	Memory Usage
LV Variables	4 bytes per element (32 bit integers) + 4 bytes
SV Variables	4 bytes per element (32 bit floating point) + 4 bytes
DV Variables	8 bytes per element (64 bit floating point) + 4 bytes
$V Variables	4 bytes + 1 byte per character
Array References	4 bytes per array reference + 4 bytes
LA Arrays	4 bytes per element + 4 bytes
SA Arrays	4 bytes per element + 4 bytes
DA Arrays	8 bytes per element + 4 bytes
$A Arrays	1 byte per character
Commands	4 bytes per command
Parametric Statements	4 bytes per operator
Long Constants	4 bytes per constant (32-bit integer)
Single Constants	4 bytes per constant (32-bit floating point)
Double Constants	8 bytes per constant (64-bit floating point)
String Constants	4 bytes + 1 byte per character
Subroutine Calls	4 bytes per level
Aliases	48 bytes per definition + 16 bytes (**#DEFINE**)
Move Buffer	136 bytes per master (**MBUF**) 108 bytes per axis 3-axis example: 136 + (3 x 108) = 460 bytes/move

Example

```
SYS>clear
SYS>dim prog0 (32768)
SYS>dim prog1 (10000)
SYS>dim def (100)
SYS>dim stream1 (8192)
SYS>dim
DIM STREAM1(8192)
DIM PROG0(32768)
DIM PROG1(10000)
DIM DEF(100)

P00>dim lv(50)
P00>dim da(2)
P00>dim da0(100)
P00>dim da1(50)
P00>dim $v(10,80)
P00>dim mbuf(10)
P00>dim
DIM DA(2)
DIM DA0(100)
DIM DA1(50)
DIM LV(50)
DIM $V(10,80)
DIM MBUF(10)
P00>
```

DIM $A — Allocate String Array Memory

Format 1	DIM $A (*count*)
Format 2	DIM $A array (*number*, *length*)
Group	Memory Control
Units	Various
Data Type	
Default	
Prompt Level	PROGx
See Also	CLEAR, MEM
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Allocates memory space for string arrays.

Arguments

Format 1

count	Specifies the number of arrays to create.

Format 2

array	Specifies the array number.
number	Specifies the number of strings to create.
length	Specifies the maximum number of characters each string can hold.

Remarks

Allocating array memory requires two steps. First, create the number of arrays you need (Format 1). This allocates memory for the arrays and creates a table of array references. Second, for each array, specify the number of variables it contains (Format 2).

Use the **DIM** command to display the current allocation.

Example

The following allocates memory space for two string arrays.

```
DIM $A(2)     : REM Dimension 2 string arrays
DIM $A0 (10,80)     : REM dimension string array 0 for 10 strings of up to
                    : REM 80 characters each
DIM $A1 (25,40)     : REM dimension string array 1 for 25 strings of up to
                    : REM 40 characters each
```

DIM $V Allocate String Memory

Format	DIM $V (*count*, *length*)
Group	Memory Control
Units	Various
Data Type	
Default	
Prompt Level	PROGx
See Also	CLEAR, MEM
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Allocates memory space for strings (8-bit character).

Arguments

count	Specifies the number of variables of this type required for the program.
length	Specifies the maximum number of characters each string can hold.

Remarks

Strings can contain any ASCII characters including control characters (characters you cannot see). See the **ASC** command for an ASCII table.

Strings longer than the dimensioned length will be truncated. String variables are local to a program and cannot be shared across programs.

Example

The following example allocates memory for five strings ($V0-$V4), with a maximum character length of 20.

```
PROGRAM
 DIM $V(5,20)
 $V0= "Hello"
 $V1= "Parker Hannifin"
 $V2= CHR$(13)+CHR$(10)+CHR$(13)+CHR$(10)
 $V3= "**********"
 $V4= "*"+SPACE$(8)+"*"
 PRINT $V0,$V1
 PRINT $V3
 PRINT $V4
 PRINT $V3,$V2
 ENDP

P03>LRUN
Hello   Parker Hannifin
**********
*        *
**********


P03>
```

DIM COM1 Allocate COM1 Memory

Format	DIM COM1 (*size*)
Group	Memory Control
Units	Bytes
Data Type	
Default	
Prompt Level	SYS
See Also	CLEAR, MEM
Related Topics	Miscellaneous Parameters
Product Revision	Command and Firmware Release

Description

Allocates memory space for the COM1 buffer.

Arguments

size Specifies the number of bytes allocated to the program.

Remarks

The default memory allocation for stream buffers (FIFO, COM1, COM2, and DPCB) is 256 bytes, which is also the minimum memory allocation for stream buffers.

When the resizing of a stream buffer is complete, the controller then sets the "Re-dimensioned" flag (bit 13) in the Stream Flags.

NOTE: To return a stream buffer to factory settings, issue the **CLEAR** command.

NOTE: The controller does not display the default memory. It only reports back memory once you have re-dimensioned it.

Example

The following illustrates allocating 4096 bytes for COM1:

```
DIM COM1(4096)
```

DIM COM2 Allocate COM2 Memory

Format	DIM COM2 (*size*)
Group	Memory Control
Units	Bytes
Data Type	
Default	
Prompt Level	SYS
See Also	CLEAR, MEM
Related Topics	Miscellaneous Parameters
Product Revision	Command and Firmware Release

Description

Allocates memory space for the COM2 buffer.

Arguments

size	Specifies the number of bytes allocated to the program.

Remarks

The default memory allocation for stream buffers (FIFO, COM1, COM2, and DPCB) is 256 bytes, which is also the minimum memory allocation for stream buffers.

When the resizing of a stream buffer is complete, the controller then sets the “Re-dimensioned” flag (bit 13) in the Stream Flags.

NOTE: To return a stream buffer to factory settings, issue the **CLEAR** command.

NOTE: The controller does not display the default memory. It only reports back memory once you have re-dimensioned it.

DIM DA Allocate Double Array Memory

Format 1	DIM DA (*number*)
Format 2	DIM DA array (*count*)
Group	Memory Control
Units	Bytes
Data Type	
Default	
Prompt Level	PROGx
See Also	CLEAR, MEM
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Allocates memory space for double array references and double arrays.

Arguments

Format 1

number	Specifies the number of arrays to create.

Format 2

array	Specifies the array number.
count	Specifies the number of variables in the array.

Remarks

Allocating array memory requires two steps. First, create the number of arrays you need (Format 1). This allocates memory for the arrays and creates a table of array references. Second, for each array specify the number of variables it contains (Format 2).

DIM DEF Allocate Defined Variable Memory

Format	DIM DEF
Group	Memory Control
Units	
Data Type	
Default	
Prompt Level	SYS
See Also	CLEAR, MEM
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Allocates memory space for defined variables.

Remarks

NOTE: Required for lineless program mode, or when using new commands (AcroBASIC version 1.18.07 or greater) in line program mode.

DIM DPCB Allocate DPCB Memory

Format	DIM DPCB (*size*)
Group	Memory Control
Units	Bytes
Data Type	
Default	
Prompt Level	SYS
See Also	CLEAR, MEM
Related Topics	Miscellaneous Parameters
Product Revision	Command and Firmware Release

(Version 1.18.06 & Up –ACR8020, ACR1505 Only).

Description

Allocates memory space for the DPCB buffer.

Arguments

size Specifies the number of bytes allocated to the program.

Remarks

The default memory allocation for stream buffers (FIFO, COM1, COM2, and DPCB) is 256 bytes, which is also the minimum memory allocation for stream buffers.

When the resizing of a stream buffer is complete, the controller then sets the "Re-dimensioned" flag (bit 13) in the Stream Flags.

To return a stream buffer to factory settings, issue the **CLEAR** command. For more information, see the CLEAR command.

DIM DV Allocate Double Variable Memory

Format	DIM DV (*count*)
Group	Memory Control
Units	Bytes
Data Type	
Default	
Prompt Level	PROGx
See Also	CLEAR, MEM
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Allocates memory space for double variables (64-bit floating point).

Argument

count	Specifies the number of variables of this type required for the program.

DIM FIFO Allocate FIFO Memory

Format	DIM FIFO (*size*)
Group	Memory Control
Units	Bytes
Data Type	
Default	
Prompt Level	SYS
See Also	CLEAR, MEM
Related Topics	Miscellaneous Parameters
Product Revision	Command and Firmware Release

(Version 1.17.03 & Up –ACR1500, ACR2000, ACR8000, ACR8010 only)

Description

Allocates memory space for FIFO.

Arguments

size	Specifies the number of bytes allocated to the program.

Remarks

The default memory allocation for stream buffers (FIFO, COM1, COM2, and DPCB) is 256 bytes, which is also the minimum memory allocation for stream buffers.

When the resizing of a stream buffer is complete, the controller then sets the "Re-dimensioned" flag (bit 13) in the Stream Flags.

NOTE: To return a stream buffer to factory settings, issue the **CLEAR** command. For more information, see the CLEAR command.

DIM LA Allocate Long Array Memory

Format 1	DIM LA (*number*)
Format 2	DIM LA array (*count*)
Group	Memory Control
Units	Bytes
Data Type	
Default	
Prompt Level	PROGx
See Also	CLEAR, MEM
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Allocates memory space for long array references and long arrays.

Arguments

Format 1

number	Specifies the number of arrays to create.

Format 2

array	Specifies the array number.
count	Specifies the number of variables in the array.

Remarks

Allocating array memory requires two steps. First, create the number of arrays you need (Format 1). This allocates memory for the arrays and creates a table of array references. Second, for each array specify the number of variables it contains (Format 2).

DIM LOGGING Allocate Logging Memory

Format	DIM LOGGING (*size*)
Group	Memory Control
Units	Bytes
Data Type	
Default	
Prompt Level	SYS
See Also	CLEAR, MEM
Related Topics	N/A
Product Revision	Command and Firmware Release

(Version 1.18 & Up – ACR1200, ACR1500, ACR1505, ACR2000, ACR8010, ACR8020 only).

Description

Allocates memory space for non-volatile, battery backed-up memory for logging parameters.

Argument

size	Specifies the number of bytes allocated to the program.

Remarks

If you allocate memory for logging parameters, they are stored in non-volatile, battery-backed memory (available for the ACR1200, ACR1500, ACR1505, ACR2000, ACR8010, ACR8020, and ACR9000 controllers).

The default, the logging parameters are stored in system memory (P20480-P20487), and this data is lost when you cycle or remove power to the controller.

DIM LV Allocate Long Variable Memory

Format	DIM LV (*count*)
Group	Memory Control
Units	Bytes
Data Type	
Default	
Prompt Level	PROGx
See Also	CLEAR, MEM
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Allocates memory space for long variables (32-bit integers).

Argument

count	Specifies the number of variables of this type required for the program.

Example

```
PROGRAM
CLEAR : REM clear local memory allocation
DIM LV(4) : REM dimension 3 long variables
LV0=12
LV1=15
LV2=LV1-LV0
LV3=10
JOG VEL X(LV0) Y (LV1) Z (LV2)  : REM assign velocities using variables
JOG FWD X Y Z  : REM start jogging on 3 axes
DWL (LV3) : REM dwell for value assigned in LV3, 10 seconds
JOG OFF X Y X
ENDP
```

DIM P Allocate Global Variable Memory

Format	DIM P (*count*)
Group	Memory Control
Units	Bytes
Data Type	
Default	
Prompt Level	SYS
See Also	CLEAR, MEM
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Allocates memory space for global variables (64-bit floating point).

Arguments

count	Specifies the number of variables of this type required for the program.

Remarks

Parameters P0-P4095 are available to users. If you designate `DIM P100`, then P0-P99 are available for your use.

NOTE: To reallocate memory space, first erase the program(s) (**NEW**), then erase the memory allocation (**CLEAR**). For more information, see the NEW and CLEAR commands.

DIM PLC Allocate PLC Memory

Format	DIM PLC plcnum (*size*)
Group	Memory Control
Units	Bytes
Data Type	
Default	
Prompt Level	SYS
See Also	CLEAR, MEM
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Allocates memory space for PLC programs.

Arguments

plcnum	Specifies the PLC number being allocated.
size	Specifies the number of bytes allocated to the program.

Remarks

NOTE: To reallocate memory space, first erase the program(s) (**NEW**), then erase the memory allocation (**CLEAR**). For more information, see the NEW and CLEAR commands.

DIM PROG Allocate Program Memory

Format	DIM PROG prognum (*size*)
Group	Memory Control
Units	Bytes
Data Type	
Default	
Prompt Level	SYS
See Also	CLEAR, MEM
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Allocates memory space for programs.

Arguments

plcnum	Specifies the PLC number being allocated.
size	Specifies the number of bytes allocated to the program.

Remarks

The default memory allocation is eight blocks of 16k×8 bytes, one block per program for programs 0 through 7.

NOTE: To reallocate memory space, first erase the program(s) (**NEW**), then erase the memory allocation (**CLEAR**). For more information, see the NEW and CLEAR commands.

To calculate the size of a program, use the following guidelines:

Data/Program Structure	**Memory Usage**
Commands	4 bytes per command
Double Constants	8 bytes per constant (64 bit floating point)
Long Constants	4 bytes per constant (32 bit integer)
Parametric Statements	4 bytes per operator
Single Constants	4 bytes per constant (32 bit floating point)
String Constants	4 bytes + 1 byte per character
Subroutine Calls	4 bytes per level

Variables	**Memory Usage**
DV Variables	8 bytes per element (64 bit floating point)
LV Variables	4 bytes per element (32 bit integers)
SV Variables	4 bytes per element (32 bit floating point)
$V Variables	4 bytes + 1 byte per character

Arrays	Memory Usage
Array References	4 bytes per array reference + 4 bytes
$A Arrays	1 byte per character
DA Arrays	8 bytes per element + 4 bytes
LA Arrays	4 bytes per element + 4 bytes
SA Arrays	4 bytes per element + 4 bytes

DIM SA Allocate Single Array Memory

Format 1	DIM SA (*number*)
Format 2	DIM SA array (*count*)
Group	Memory Control
Units	Bytes
Data Type	
Default	
Prompt Level	PROGx
See Also	CLEAR, MEM
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Allocates memory space for single array references and single arrays.

Arguments

Format 1

number	Specifies the number of arrays to create.

Format 2

array	Specifies the array number.
count	Specifies the number of variables in the array.

Remarks

Allocating array memory requires two steps. First, create the number of arrays you need (Format 1). This allocates memory for the arrays and creates a table of array references. Second, for each array specify the number of variables it contains (Format 2).

DIM STREAM Allocate STREAM Memory

Format	DIM STREAM *stream* (*size*)
Group	Memory Control
Units	Bytes
Data Type	
Default	
Prompt Level	SYS
See Also	CLEAR, MEM
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Allocates memory space for STREAM.

Arguments

stream

size

Remarks

The default memory allocation for stream buffers (FIFO, COM1, COM2, and DPCB) is 256 bytes, which is also the minimum memory allocation for stream buffers.

When the resizing of a stream buffer is complete, the controller then sets the "Re-dimensioned" flag (bit 13) in the Stream Flags.

NOTE: To return a stream buffer to factory settings, issue the **CLEAR** command. For more information, see the CLEAR command.

DIM SV Allocate Single Variable Memory

Format	DIM SV (*count*)
Group	Memory Control
Units	Bytes
Data Type	
Default	
Prompt Level	PROGx
See Also	CLEAR, MEM
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Allocates memory space for single variables (32-bit floating point).

Argument

count	Specifies the number of variables of this type required for the program.

DIN Dead Zone Integral Initial Negative Value

Format	DIN {*axis* {*value*}} {*axis* {*value*}} ...
Group	Servo Control
Units	Volts
Data Type	FP32
Default	0
Prompt Level	PROGx
See Also	AXIS, DIP, DZL, DZU
Related Topics	Axis Parameters (0-7) (8-15)
Product Revision	Command and Firmware Release

This command sets the dead zone integral initial negative value of an axis. Each time the servo loop comes out of dead zone with a negative following error, the Integrator of the PID loop gets the **DIN** value as its initial value. Issuing a **DIN** command to an axis without an argument will display the current setting for that axis. The default value is 0.

Example

The following example sets the X axis **DIN** value to –1.5 volt.

```
DIN X -1.5
```

DIP Dead Zone Integral Initial Positive Value

Format	DIP {*axis* {*value*}} {*axis* {*value*}} ...
Group	Servo Control
Units	Volts
Data Type	FP32
Default	0
Prompt Level	PROGx
See Also	AXIS, DZL, DZU, DIN
Related Topics	Axis Parameters (0-7) (8-15)
Product Revision	Command and Firmware Release

This command sets the dead zone integral initial positive value of an axis. Each time the servo loop comes out of dead zone with a positive following error, the Integrator of the PID loop gets the **DIP** value as its initial value. Issuing a **DIP** command to an axis without an argument will display the current setting for that axis. The default value is 0.

Example

The following example sets the X axis **DIP** value to 2.2 volt.

```
DIP X2.2
```

DRIVE Drive Report-back

Format	DRIVE *command* {*axis*} {*axis*}...
Group	Drive Control
Units	Axis number
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	AXIS, DRIVE OFF, DRIVE ON, DRIVE RES
Related Topics	N/A
Product Revision	Command and Firmware Release

The **DRIVE** command provides a report-back of the drive's status, lets you energize/de-energizes a motor/drive combination or reset the drive.

The following is a list of valid command combinations:

DRIVE: Displays current configuration.

DRIVE ON: Enables the drive.

DRIVE OFF: Disables the drive.

DRIVE RES: Resets the drive.

Example

The following example reports the drive status of axis X.

```
DRIVE X
DRIVE OFF
```

DRIVE OFF Drive Disable

Format	DRIVE OFF {*axis*} {*axis*}...
Group	Drive Control
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	AXIS, DRIVE, DRIVE ON, DRIVE RES
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

The **DRIVE OFF** command de-energizes a motor/drive combination. This is the same as clearing the Drive Enable Output (DEO) bit in the Quaternary Axis Flags.

Remarks

DRIVE commands are only valid for ACR90x0 and ARxxCE products.

Issuing a **DRIVE OFF** command while an axis is moving (**JOG**, **CAM**, **GEAR**, or coordinated moves) will result in a `NOT VALID WHILE IN MOTION` error message.

The **DRIVE OFF** function can be executed from an HMI or PC application by clearing the Drive Enable Output flag, found in Bit Index 17 of the Quaternary Axis Flags.

- Axis 0: CLR8465
- Axis 1: CLR8497

Examples

Example 1

The following example disables the drive on axis X.

```
DRIVE OFF X
```

Example 2

Multiple axes can be designated on one line.

```
DRIVE OFF X Y
```

Example 3

Using the Axis name “X” is only valid in the program where the axis is attached. The following syntax is valid in any program or program prompt:

```
AXIS0 DRIVE OFF

DRIVE OFF AXIS0

DRIVE OFF AXIS0 AXIS1
```

Example 4

It is also possible to use a variable to indicate the Axis number.

```
DIM LV(1)
REM this loop will enable all drives
FOR LV0 = 0 TO 15 STEP 1
  DRIVE OFF AXIS(LV0)
  NEXT
```

DRIVE ON Drive Enable

Format	DRIVE ON {*axis*} {*axis*}...
Group	Drive Control
Units	Axis number
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	AXIS, DRIVE, DRIVE OFF, DRIVE RES
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

The **DRIVE ON** command energizes a motor/drive combination. This is the same as setting the Drive-Enable Output bit in the Quaternary Axis Flags.

Remarks

The **DRIVE ON** function can be executed from an HMI or PC application by setting the Drive Enable Output flag, found in Bit Index 17 of the Quaternary Axis Flags.

Examples

Example 1

The following example enables the drive on axis X.

```
DRIVE ON X
```

Example 2

Multiple Axes can be designated on one line.

```
DRIVE ON X Y
```

Example 3

Using the Axis name "X" is only valid in the program where the axis is attached. The following syntax is valid in any program or program prompt.

```
AXIS0 DRIVE ON

DRIVE ON AXIS0

DRIVE ON AXIS0 AXIS1
```

Example 4

It is also possible to use a variable to indicate the axis number.

```
DIM LV(1)
REM this loop will enable all drives
FOR LV0 = 0 TO 15 STEP 1
  DRIVE ON AXIS(LV0)
  NEXT
```

DRIVE RES Drive Reset

Format	DRIVE RESET {*axis*} {*axis*}...
Group	Drive Control
Units	Axis number
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	AXIS, DRIVE, DRIVE OFF, DRIVE ON
Related Topics	N/A
Product / Rev Notes	ACR90x0, ARxxCE only

Description

The **DRIVE RESET** command performs a hardware reset for certain motor/drive combinations. This is the same as setting the Drive-Reset Output bit in the Quaternary Axis Flags.

Remarks

Issuing a **DRIVE RES** command while an axis is moving (**JOG**, **CAM**, **GEAR**, or coordinated moves) will result in a `NOT VALID WHILE IN MOTION` error message.

Examples

Example 1

Resets the drive on axis X.

```
DRIVE RES X
```

Example 2

Multiple Axes can be designated on one line.

```
DRIVE RES X Y
```

Example 3

Using the Axis nickname “X” is only valid in the program where the axis is attached. The following syntax is valid in any program or program prompt.

```
AXIS0 DRIVE RES

DRIVE RES AXIS0

DRIVE RES AXIS0 AXIS1
```

Example 4

It is also possible to use a variable to indicate the axis number.

```
DIM LV(1)
REM this loop will reset all drives
FOR LV0 = 0 TO 15 STEP 1
  DRIVE RES AXIS(LV0)
  NEXT
```

DTALK Drive Talk

Format	DTALK {*axis*} {*axis*}...
Group	Character I/O
Units	Axis number
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	CLOSE, OPEN DTALK
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

The **DTALK** command directs subsequent commands to a specified axis. Before you can use the **DTALK** command, you must open communication with a device—use the **OPEN DTALK** command. In addition, you can use the **CLOSE** command to end communications with the device.

Remarks

You can also communicate directly with a drive that has the Drive Talk feature. The controller acts as a pass through, and communication does not affect the controller. You can use this method for troubleshooting drive and controller problems.

When directly communicating with a drive featuring Drive Talk, be aware of the following:

- To exit direct communication through the terminal, you must send an escape character.
- Use the **LRUN** command to run the program.
- When you open a drive talk stream, you must assign it a device number. Then assign the same device number to the Communication Device parameter for the axis to which you want to connect. See Example 2 below.
- You must declare what type of drive you are using with the Drive Type parameter.

NOTE: The **DTALK** command only works with controllers and drives that have the Drive Talk feature. You must use the 71-021599-xx Aries Drive cable.

Examples

Example 1

The following example sets up drive talk for two axes. It then retrieves the encoder position for each axis and prints the data.

```
OPEN DTALK "COM2:9600,N,8,1" AS #1 REM OPEN PORT
P28672=1    : REM SET COMMUNICATION DEVICE NUMBER FOR DRIVE 1
P28928=1    : REM SET COMMUNICATION DEVICE NUMBER FOR DRIVE 2
P28673=0    : REM SET DRIVE TYPE FOR AXIS1 TO ARIES DRIVES
P28929=0    : REM SET DRIVE TYPE FOR AXIS2 TO ARIES DRIVES
CLR 11122    : REM RESET TIMEOUT
CLR 11123    : REM RESET TIMEOUT
CLR 11124    : REM RESET TIMEOUT
SET 10505    : REM GET TPE AXIS0 USING GET DRIVE DATA
SET 10500    : REM UPDATE DATA AXIS0 USING

REM GET_DRIVE_DATA_REQUEST
SET 10537    : REM GET TPE AXIS1 USING GET DRIVE DATA
SET 10532    : REM UPDATE DATA AXIS1 USING

REM GET_DRIVE_DATA_REQUEST
?P28693    : REM SHOW TPE AXIS0 ON TERMINAL
?P28949    : REM SHOW TPE AXIS1 ON TERMINAL

SET 10500    : REM GET TPE AXIS1 USING GET DRIVE DATA
SET 10532    : REM UPDATE DATA AXIS1 USING

REM GET_DRIVE_DATA_REQUEST
?P28693    : REM SHOW TPE AXIS0 ON TERMINAL
?P28949    : REM SHOW TPE AXIS1 ON TERMINAL
```

Example 2

You can use the following program to directly communicate with an Aries drive in a terminal. The program is designed for AXIS0 and PROG0, though you can change it as necessary.

```
OPEN DTALK "COM2:9600,N,8,1" AS #1
REM OPEN A DRIVE TALK PORT WITH DEVICE NUMBER 1
P28672=1
REM SET COMMUNICATION DEVICE, AXIS 0, TO DEVICE NUMBER 1
REM THE DEVICE NUMBER MUST MATCH THE OPEN COMMAND ABOVE
P28673=0    : REM SET AXIS0 TO AN ARIES DRIVE
CLR11122 CLR11123 CLR11124    : REM CLEAR ALL TIMEOUT BITS
SET11104    : REM START AUTO ADDRESS
DTALK X    : REM START TALKING DIRECTLY TO THE DRIVE
ENDP
```

Drive Talk for EPL

The Drive Talk for EPL feature is basically the same as Dive Talk for Aries, except that auto-addressing is not used, the "Send Error Log" request is not used, and the **OPEN DTALK** command is not required. The controller performs an automatic **OPEN** and **CLOSE** for each axis requesting a Drive Talk exchange. To use Dive Talk for EPL, the user only needs to set the desired Dive Talk axis control flag.

The EPL network must be operational (**EPLC ON**) to use this feature. No EPLD streams may be currently "OPEN" when the Dive Talk requests are made. The node status of the node corresponding to the Dive Talk axis must be "Node Found."

A Communication Device number between 1 and 7 inclusive must be entered in the Drive Configuration Parameters table for each axis that will use Dive Talk. The same number may be used for all axes, but this number must not already be in use as the device number for another "OPEN" stream. Because most of the examples in this **DTALK** section refer to device #1, a value from 2 to 7 may be best.

For an Aries EPL drive, set the Drive Type Parameter to 1.

The controller will wait until all of these requirements are met before responding to one of the Drive Talk request bits.

The controller signals that the request has been completed or failed by automatically clearing the request bit. If the controller was not able to make the Dive Talk connection, the EPL network status bit 16653 will be set. If a timeout (100 milliseconds) occurred, bit 11059 of the EPL Drive Talk flags will be set. This latter bit must be cleared by the user before another attempt at Dive Talk will succeed.

To send or receive data, use the following Drive Control Flags:

- Get Config Request
- Send Config Request
- Get Drive Data Request

When the Get Config Request is successful, values are populated in the Drive Configuration Parameters. The Get Config Request flag will clear when the request is complete.

When the Get Drive Data Request is successful, values are populated in the Drive Talk Status Parameters. The Get Drive Data Request flag will clear when the request is complete.

DWIDTH Set Derivative Sample Period

Format	DWIDTH {*axis* {*value*}} {*axis* {*value*}} ...
Group	Servo Control
Units	seconds
Data Type	FP32
Default	0
Prompt Level	PROGx
See Also	AXIS, DGAIN, PERIOD
Related Topics	Axis Parameters (0-7) (8-15)
Product Revision	Command and Firmware Release

This command modifies the value used in the PID algorithm to control the derivative sampling rate. Issuing a **DWIDTH** command to an axis without an argument will display the current setting for that axis. The default width is 0.0 for all axes.

Derivative sampling width determines how often the following error is sampled when calculating the derivative term. Setting this value to zero will set the sampling to occur at the servo interrupt rate set with the **PERIOD** command.

Example

The following example sets the X axis derivative sample width to 1 millisecond:

```
DWIDTH X0.0001
```

DWL Delay for a Given Period

Format	DWL *time*
Group	Logic Function
Units	seconds
Data Type	FP32
Default	N/A
Prompt Level	PROGx
See Also	PROG
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Suspends program execution for a given amount of time.

Remarks

The minimum dwell time is 1 millisecond.

NOTE: Use caution when using dwell statements in non-motion programs (PROG8 through 15 *). A dwell in one non-motion program will cause a dwell for all non-motion programs because they share a processor time slice.

* PROG1 through 15 for Aries Controller.

Example

The following example will delay for 1.25 seconds.

```
DWL 1.25
```

DZL Dead Zone Lower Limit

Format	DZL {*axis* {*value*}} {*axis* {*value*}}....
Group	Servo Control
Units	pulses
Data Type	Long
Default	0
Prompt Level	PROGx
See Also	AXIS, DZU, DIP, DIN
Related Topics	Axis Parameters (0-7) (8-15), Tertiary Axis Flags (0-7) (8-15)
Product Revision	Command and Firmware Release

This command sets the lower limit for the dead zone of an axis. The **DZL** value should be less than **DZU**. Issuing a **DZL** command to an axis without an argument will display the current setting for that axis. The default value is 0.

Once the current commanded position of the axis equals the target position of the axis, the dead zone mechanism becomes active. Then as soon as the following error becomes less the **DZL**, the DAC output goes to zero and stays there until the following error becomes greater than **DZU**.

Example

The following example sets the X axis dead zone lower limit to 5 pulses.

```
DZL X 5
```

DZU Dead Zone Upper Limit

Format	DZU {*axis* {*value*}} {*axis* {*value*}} ...
Group	Servo Control
Units	Pulses
Data Type	Long
Default	0
Prompt Level	PROGx
See Also	AXIS, DZL, DIP, DIN
Related Topics	Axis Parameters (0-7) (8-15)
Product Revision	Command and Firmware Release

This command sets the upper limit for the dead zone of an axis. The axis will remain in the dead zone with DAC output of zero, until the following error becomes greater than **DZU**. Note that the **DZU** value should be greater than **DZL**. Issuing a **DZU** command to an axis without an argument will display the current setting for that axis. The default value is 0.

Example

The following example sets the X axis dead zone upper limit to 12 pulses.

```
DZU X 12
```

ECHO Control Character Echoing

Format	ECHO {*mode*}
Group	Operating System
Units	N/A
Data Type	N/A
Default	1
Prompt Level	SYS, PROGx, PLCx
See Also	CLOSE, OPEN
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Controls the prompt and echo on a communication channel.

Argument

mode

Remarks

Issuing an **ECHO** command without an argument displays the current setting.

The default echo mode is enabled (**ECHO1**) for all communication channels. This command affects only the port from which it is issued. However, if you set the echo mode to zero (**ECHO0**) download is faster.

NOTE: While downloading user and PLC programs, you can view the progress, line by line, in the terminal. In the terminal you can see the report back of download errors.

The following table lists the valid echo modes:

Echo Mode	Command Prompt	Error Messages	Character Echo
0	ON	ON	OFF
1	ON	ON	ON
2	ON	OFF	OFF
3	ON	OFF	ON
4	OFF	ON	OFF
5	OFF	ON	ON
6	OFF	OFF	OFF
7	OFF	OFF	ON

Example

The following turns off error message reporting:

```
ECHO 3
```

ELOAD Load System Parameters

Format	ELOAD {ALL}
Group	Nonvolatile
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	BRESET, ESAVE, ERASE, FLASH, PBOOT
Related Topics	N/A
Product Revision	Command and Firmware Release

This command loads the system parameters that were stored in EEPROM (ACR8000 only) or **FLASH** system parameter section (for all other boards) using the **ESAVE** command.

Note that the "ALL" command modifier is optional.

The values loaded with **ELOAD** include:

- System attachments
- Master parameters:

  ACC, **DEC** and **STP** ramps

  VEL, **FVEL** and **IVEL** values
- Axis parameters:

  Gain and limit settings

  PPU and **VECDEF** values

  ON / OFF states
- Encoder multipliers
- **DAC** gains and offsets
- **ADC** mode, gains and offsets

NOTE: Program memory allocation is stored in battery-backup RAM, not in the EEPROM.

NOTE: **FLASH** commands (**FLASH SAVE**, **FLASH ERASE**, etc.) have no effect on the saved system parameters.

Example

```
ELOAD
```

ENC Direct ENC Manipulation

Format 1:	ENC *index* RES {*preload*}
Format 2:	ENC *index* MULT {*multiplier*}
Group	Global Objects
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	AXIS, DAC, MULT, RES
Related Topics	N/A
Product Revision	Command and Firmware Release

The **ENC** commands give direct access to encoder reset and multiplier setup without going through the axes. Issuing these commands without the final argument will display their current settings. See the corresponding base commands for descriptions:

Example

The following example sets the hardware for **ENC5** to 4x multiplication:

```
ENC5 MULT 4
```

ENC CLOCK SSI Encoder Clock Frequency

Format	ENC *index* CLOCK {*mode*}
Group	Global Object (Encoder)
Units	N/A
Data Type	Long
Default	0
Prompt Level	SYS, PROGx, PLCx
See Also	ENC
Related Topics	N/A
Product Revision	Command and Firmware Release

This command sets the frequency of the clock signal that is generated by the FPGA and given to the SSI encoder. In response the SSI encoder will transmitted back the position data at the same frequency. Typically the encoder manufactures rate their encoder as capable of handling a frequency of up to certain limit. Any frequency lower than this limit should work fine as well.

NOTE: The frequency should be fast enough so that the whole data safely reaches ACR9000 before the next servo period.

ENC2 CLOCK 1 will set the clock frequency to 781.2 KHz.

Mode	Clock Frequency (KHz)
0	1562.5
1	781.2
2	390.6
3	195.3
4	97.6

Example

```
SYS> ENC2 CLOCK 1
SYS> ENC2 CLOCK
13
SYS>

ENC2 CLOCK 1
ENC2 DST 0
ENC2 SRC 3
```

ENC DST SSI Encoder Data Format

Format	ENC *index* DST {*mode*}
Group	Global Object (Encoder)
Units	NA
Data Type	Long
Default	0
Prompt Level	SYS, PROGx, PLCx
See Also	ENC
Related Topics	N/A
Product Revision	Command and Firmware Release

This command is used to configure the format of serial Data Stream. A value of zero will tell the controller that the data is in GRAY code and it needs to convert is standard binary format for the firmware. Similarly setting this value to one will tell the FPGA that no conversion is necessary, and the data can just pass through.

Mode	Data Format
0	Gray Code
1	Standard Binary

Example

```
SYS> ENC2 DST 1
SYS> ENC2 DST
1
SYS>

ENC2 CLOCK 1
ENC2 DST 0
ENC2 SRC 3
```

ENC LIMIT SSI Encoder Rollover Limit

Format	ENC *index* LIMIT {*number_of_bits*}
Group	Global Object (Encoder)
Units	Number of BITS
Data Type	Long
Default	31 BITS
Prompt Level	SYS, PROGx, PLCx
See Also	ENC
Related Topics	N/A
Product Revision	Command and Firmware Release

This command sets the number of bits at which the SSI data rollover will occur. This number is always less than or equal to **ENC WIDTH**. For most encoders this number should be same as the **ENC WIDTH**. In such a case there is no need to issue this command.

The range of this number is 0 and 31.

Example

```
SYS> ENC2 LIMIT 13
SYS> ENC2 LIMIT
13
SYS>
```

ENC SRC Encoder Source

Format	ENC {*axis*} SRC {*source*}
Group	Global Object (Encoder)
Units	NA
Data Type	Long
Default	0
Prompt Level	SYS, PROGx, PLCx
See Also	ENC
Related Topics	N/A
Product Revision	Command and Firmware Release

The **ENC SRC** command lets you select the type of feedback used with your encoder.

Source	Input	Channel A	Channel B
0	Quadrature	Channel A	Channel B
1	Step and Direction	Step	Direction
2	CW/CCW	CW Step	CCW Step
3	SSI	SCLK	SDATA
4	RESERVED		
5	Internal Step and Direction	Step	Direction
6	Clockwise/Counter Clockwise	CW Step	CCW Step
7	RESERVED		

Example

The following example sets the feedback source to quadrature for encoder 1.

```
ENC1 SRC0
```

ENC WIDTH SSI Encoder Data Bit Length

Format	ENC *index* WIDTH {*number_of_bits*}
Group	Global Object (Encoder)
Units	Number of BITS
Data Type	Long
Default	31 BITS
Prompt Level	SYS, PROGx, PLCx
See Also	ENC
Related Topics	N/A
Product Revision	Command and Firmware Release

This command sets the number of bits in the serial data transmitted by the SSI encoder. The range of this number is 0 and 31.

Example

```
SYS> ENC2 WIDTH 13
SYS> ENC2 WIDTH
13
SYS>
```

END End of Program Definition

Format	END
Group	Program Flow
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	N/A
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Causes a program to terminate.

Remarks

The **END** command is used to terminate the program in the middle based on some condition. Issuing an **END** command from the command line will not stop the execution of a program, use the **HALT** command instead.

If the program executes to the last line, an **END** command is automatically done.

Example

The following example ends program execution when bit 10 is TRUE.

```
_Loop1
IF (BIT10) THEN END
GOTO LOOP1
```

ENDP End Line-numberless Program

Format	ENDP
Group	Program Flow
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	GOSUB, GOTO, PROGRAM, RUN
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Marks the end of a program that does not use line numbers.

Remarks

The **PROGRAM** command marks the start of a program that does not use line numbers.

If the **ENDP** command is not included, then immediate mode commands are not executed. Instead, the commands are stored in the program space as well.

NOTE: When using the **LIST** command to view a lineless program, you can view the automatic line numbers by setting bit 5651. (**SET 5651**)

Example

```
SYS
HALT ALL
NEW ALL
CLEAR
DIM PROG0(1000)
DIM DEF 10
#DEFINE LED BIT96
#DEFINE myflag BIT 32
#DEFINE TRUE 1

PROG0
#DEFINE Counter LV2
#DEFINE loop LV4

PROGRAM
DIM LV10
Counter=0
SET myflag
WHILE (myflag)
    Counter= Counter+1
    FOR loop=100 TO 500 STEP 200
        PRINT loop ;",";
    NEXT
    GOSUB SetLED
    IF (Counter>5)
        CLR myflag
        PRINT " Done "
        ELSE
        PRINT " Busy "
    ENDIF
WEND
```

```
PRINT "END OF PROGRA,"
END

_SetLED
SET LED
PRINT " LED on "; LED,
RETURN

ENDP
```

EPLC Define EPLC

Format	EPLC
Group	Drive Control
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	DIAG EPLD, EPLC OFF, EPLC ON, OPEN EPLD
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Reports ETHERNET Powerlink (EPL) network and node status.

Remarks

This is for a network managed with an **EPLC** (ETHERNET Powerlink controller).

Example

```
SYS>EPLC
Number of Controlled nodes (P37376) = 3
EPL network state = 5 (EPL operational)
current operational duration = 86 (0:1:26)
EPLD0 Node ID 1 Node status = 1
EPLD1 Node ID 2 Node status = 1
EPLD2 Node ID 3 Node status = 1
```

EPLC OFF Reset ETHERNET Powerlink Network

Format	EPLC OFF
Group	Drive Control
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	DIAG EPLD, EPLC, EPLC_ON, OPEN EPLD
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Resets the ETHERNET Powerlink (EPL) network.

Remarks

For safety reasons, the user may not reset the EPL network while any ETHERNET Powerlink drives (EPLD) are enabled.

In certain instances after an **EPLC OFF** is issued, it may take up to one minute to reacquire the network after an **EPLC ON** is issued.

Example

```
SYS>EPLC OFF
```

EPLC ON Start ETHERNET Powerlink Network

Format	EPLC ON
Group	Drive Control
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	DIAG EPLD, EPLC, EPLC OFF, OPEN EPLD
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Starts the ETHERNET Powerlink (EPL) network.

Remarks

For safety reasons, the user may not start the EPL network while any ETHERNET Powerlink drives (EPLD) are enabled. Also, the EPL network may not be started if the trajectory period (as specified with the **PERIOD** command) is too low for the number of nodes on the network. (Minimum period in microseconds = 125 x (number of nodes) with some restrictions. See *ACR9000 Hardware Installation Guide*, Appendix H, for more information.)

When the **EPLC ON** command is sent, the ACR9030/9040 initially places all EPLDs into the "Pre-Operational" state, during which the controller interrogates and configures the EPLDs as required. The EPLDs are then placed into the "Operational" state, and the automatic data transfer between the EPLDs and the ACR9030/9040 takes place.

In certain instances after an **EPLC OFF** is issued, it may take up to one minute to reacquire the network after an **EPLC ON** is issued.

Examples

Example 1

```
SYS>EPLC ON
```

Example 2

Using EPLC with a PBOOT program. The controller takes a finite amount of time after startup to register the EPL controller card. Therefore, it is a good idea to wait until the EPL card is available. The following program demonstrates this.

```
PROGRAM
PBOOT   : REM program starts on controller power cycle
INH 16648    : REM wait for controller to detect EPL card
IF (NOT BIT 16649)    : REM check if EPL network operational
       EPLC ON   : REM Turns on EPL Network. Same as SET 16640
       REM loop until Network Start either failed or successful
       WHILE (NOT(BIT 16650 OR BIT 16649))
       WEND
       ENDIF
ENDP
```

EPLD Direct EPLD Manipulation

Format 1:	EPLD *index* RES {*preload*}
Group	Global Objects
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	AXIS, DAC, ENC, MULT, RES
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Gives direct access to EPLD reset without going through the axes.

Example

The following sets the EPLD5 position to 1000:

```
SYS>EPLD5 RES 1000
```

ERASE Erase the System Parameters

Format	ERASE {ALL}
Group	Nonvolatile
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	BRESET, ELOAD, ESAVE, FLASH, PBOOT
Related Topics	N/A
Product Revision	Command and Firmware Release

This command erases all system parameter information from the EEPROM (ACR8000) or flash system parameter section (for all other boards). When you next cycle power, the controller uses the factory default settings instead of the EEPROM or flash values.

The **ALL** command modifier is optional.

NOTE: This command should be used with caution since it destroys setup information.

NOTE: Program memory allocation is stored in battery-backup RAM, not in the EEPROM.

NOTE: **FLASH** commands (**FLASH SAVE**, **FLASH ERASE**, etc.) have no effect on the system parameters.

NOTE: The following commands issued at the SYS> prompt will completely clear the memory on all cards, except ACR8000. The order of commands is crucial.

```
HALT ALL
NEW ALL
DETACH ALL
CLEAR
ERASE
FLASH ERASE (omit for ACR8000)
CONFIG CLEAR
CLEAR FIFO (omit for ACR9000)
CLEAR COM1 (CLEAR DPCB for ACR1505, ACR8020)
CLEAR COM2
BRESET
REBOOT
```

Example

The following example erases the EEPROM or flash system parameters, depending on which controller you issue the command.

```
ERASE
```

ESAVE Save System Parameters

Format	ESAVE {ALL}
Group	Nonvolatile
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	ELOAD, ERASE, PBOOT, BRESET
Related Topics	N/A
Product Revision	Command and Firmware Release

This command stores system parameters into EEPROM (ACR8000) or flash system parameter section (for all other boards) to be retrieved on power-up or by issuing an **ELOAD** command.

Note that the **ALL** command modifier is optional.

The values stored by **ESAVE** include:

- System attachments
- Master parameters:

  ACC, **DEC** and **STP** ramps

  VEL, **FVEL** and **IVEL** values
- Axis parameters:

  Gain and limit settings

  PPU and **VECDEF** values

  ON / OFF states
- Encoder multipliers
- **DAC** gains and offsets
- **ADC** mode, gains and offsets

NOTE: Program memory allocation is stored in battery-backup RAM, not in the EEPROM.

NOTE: **FLASH** commands (**FLASH SAVE**, **FLASH ERASE**, etc.) have no effect on the saved system parameters.

Example 1

```
ESAVE
```

Example 2

The following example provides a good example of saving all parameters and programs. Perform this only when all commanded motion is stopped.

```
HALT ALL
FLASH IMAGE
ESAVE ALL
```

EXC Set Excess Error Band

Format	EXC {*axis* {*value*}} {*axis* {(*value1*, *value2*)}} ...
Group	Axis Limits
Units	Units based on PPU
Data Type	FP32
Default	0,0
Prompt Level	PROGx
See Also	AXIS, IPB, PPU
Related Topics	Axis Flags (0-7)(8-15), Axis Parameters (0-7) (8-15), Master Flags (0-7) (8-15)
Product Revision	Command and Firmware Release

This command sets the following error limits monitored by the "not excess error" flags. When the following error of a given axis is within its excess error band, the appropriate flag is set. Otherwise, the flag is cleared. For masters, the flag is set if all of its slaves are within their excess error bands.

The setting of these flags is the only reaction the controller will take. It is up to the programmer to write code that will react to the flags in an appropriate manner.

Issuing the **EXC** command to an axis without an argument displays the current positive and negative limits for that axis. Issuing the command with a single argument sets the positive limit to *value* and the negative limit to minus *value*. Issuing the command with two arguments sets the positive limit to *value1* and the negative limit to *value2*. The default for both is 0.0 for all axes.

Example 1

The following example sets an excess error band of ±0.5 units for X and Y axes.

```
EXC X0.5 Y0.5
```

Example 2

The following example set an excess error band of +3 and –1 units for the x-axis.

```
EXC X(+3,-1)
```

F Set Velocity in Units/Minute

Format	**F** {*rate*}
Group	Velocity Profile
Units	units / minute
Data Type	FP32
Default	600
Prompt Level	PROGx
See Also	ACC, DEC, FOV, MASTER, STP, PPU, VEL
Related Topics	N/A
Product Revision	Command and Firmware Release

This command is an alternative to using the **VEL** command. The **F** command works identically to the **VEL** command except for a scaling modifier that translates the move velocity into units / minute. It also directly changes **VEL**.

Example

The following example sets the velocity to 600 units per minute (same as **VEL 10**):

```
F600
```

FBVEL Set Velocity Feedback Gain

Format	FBVEL {*axis* {*value*}} {*axis* {*value*}} …
Group	Servo Control
Units	volts / pulses / second
Data Type	FP32
Default	0
Prompt Level	PROGx
See Also	AXIS, ATTACH AXIS, DGAIN
Related Topics	Axis Parameters (0-7) (8-15)
Product Revision	Command and Firmware Release

This sets the velocity feedback gain for an axis. Issuing an **FBVEL** command to an axis without an argument will display the current setting for that axis. The default velocity feedback gain is 0.0 for all axes.

The velocity feedback gain is multiplied by the derivative (change) of the velocity feedback source attached to the axis with the **ATTACH AXIS** command. This value is then subtracted from the control signal before it enters the digital filters.

The result is a software tachometer based on encoder or analog signal input. A typical use for this would be a dual-feedback loop where an encoder on the load is used for the position feedback and an encoder on the motor shaft is used to dampen velocity response.

Example

The following example sets X axis velocity feedback gain to 0.0001 volts / pulses / second:

```
FBVEL X0.0001
```

FFACC Set Feedforward Acceleration

Format	FFACC {*axis* {*value*}} {*axis* {*value*}} ...
Group	Servo Control
Units	volts / pulses / second2
Data Type	FP32
Default	0
Prompt Level	PROGX
See Also	AXIS, FFVEL, PPU, IGAIN, DGAIN
Related Topics	Axis Parameters (0-7) (8-15)
Product Revision	Command and Firmware Release

This sets the acceleration feedforward for an axis. Issuing an **FFACC** command to an axis without an argument will display the current setting for that axis. The default acceleration feedforward gain is 0.0 for all axes.

The correct value can be determined using the following formula:

FFACC = **PGAIN** * **ERROR** / **ACCEL**

Where:

PGAIN = proportional gain (volts / pulse)

ERROR = error at a given acceleration (pulses)

ACCEL = the given acceleration (pulses / second^2)

Note that this formula only applies after the velocity feedforward gain has been set correctly with the **FFVEL** command. Otherwise, velocity errors will be present as well.

Example

The following example sets X axis acceleration feedforward to 0.000001 volts / pulses / second / second:

```
FFACC X0.000001
```

FFVC Feed Forward Velocity Cutoff Before Target

Format	FFVC {*axis* {*value*}} {*axis* {*value*}} ...
Group	Servo Control
Units	Pulses
Data Type	LONG
Default	0
Prompt Level	PROGx
See Also	AXIS, DIN, DIP, DZL, DZU
Related Topics	Axis Parameters (0-7) (8-15)
Product Revision	Command and Firmware Release

This command sets the band around the target point, in which the feed forward velocity term is made zero. Issuing a **FFVC** command to an axis without an argument will display the current setting for that axis. The default value is 0.

Example

The following example sets the X axis **FFVC** band to 100 pulses.

```
FFVC X 100
```

FFVEL Set Feedforward Velocity

Format	FFVEL {*axis* {*value*}} {*axis* {*value*}} ...
Group	Servo Control
Units	volts / pulses / second
Data Type	
Default	
Prompt Level	
See Also	AXIS, DGAIN, FFACC, IGAIN, PGAIN, PPU
Related Topics	Axis Parameters (0-7) (8-15)
Product Revision	Command and Firmware Release

This sets the velocity feedforward for an axis. Issuing an **FFVEL** command to an axis without an argument will display the current setting for that axis. The default velocity feedforward gain is 0.0 for all axes.

The correct value can be determined using the following formula:

FFVEL = **PGAIN** * **ERROR** / **VELOC**

Where:

PGAIN = proportional gain (volts / pulse)

ERROR = error at a given velocity (pulses)

VELOC = the given velocity (pulses / second)

Note that this formula will not work correctly if there is any DC offset in the drives. Either adjust the drives, or use the PID integral term (**IGAIN**)to remove the offset first..

Example

The following example sets X axis velocity feedforward to 0.0001 volts / pulses / second:

```
FFVEL X0.0001
```

FIRMWARE Firmware Upgrade/Backup (ACR8020 Only)

Format	FIRMWARE *command*
Group	Nonvolatile
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS
See Also	FIRMWARE BACKUP, FIRMWARE CHECKSUM
Related Topics	Miscellaneous Control Group 1 Flags
Product Revision	Command and Firmware Release

NOTE: This command applies only to the ACR8020 controller.

This command is used along with a second command to program new firmware into flash memory. The flash memory of ACR8020 was divided into 5 blocks: Bootflash, Sysflash1, Sysflash2, Userflash and Flashslot. Two copies of firmware code are programmed into flash memory at the factory. The Sysfalsh1 area stores the first copy of firmware code and the Sysfalsh2 area stores the second copy of firmware code. The Bootflash area stores the bootloader code, which checks the validity of Sysflash1. If Sysflash1 code is valid, it will be loaded into program RAM at power-up. Otherwise Sysflash2 code will be loaded into program RAM at power-up. The Userflash area stores user programs and parameters by using **FLASH** commands. Flashslot store system parameters by using **ESAVE** command.

The following is a list of valid firmware command combinations:

FIRMWARE UPGRADE: Program firmware code into the Sysflash1 area.

FIRMWARE BACKUP: Backup firmware code form Sysflash1 to Sysflash2.

FIRMWARE CHECKSUM: Calculate firmware checksums.

FIRMWARE BACKUP Firmware Backup (ACR8020 Only)

Format	FIRMWARE BACKUP
Group	Nonvolatile
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS
See Also	FIRMWARE UPGRADE
Related Topics	N/A
Product Revision	Command and Firmware Release

NOTE: This command applies only to the ACR8020 controller.

This command copies firmware code from Sysflash1 to Sysflash2 if the Sysflash1 code is valid. Do not power down or perform any other functions until this operation is completed.

FIRMWARE CHECKSUM Calculate Firmware Checksum (ACR8020 Only)

Format	FIRMWARE CHECKSUM
Group	Nonvolatile
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS
See Also	FIRMWARE UPGRADE
Related Topics	Miscellaneous Parameters
Product Revision	Command and Firmware Release

NOTE: This command applies only to the ACR8020 controller.

This command calculates firmware checksum. All the firmware related flags and parameters are not valid until this operation is completed.

FIRMWARE UPGRADE Firmware Upgrade

Format	FIRMWARE UPGRADE
Group	Nonvolatile
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS
See Also	FIRMWARE CHECKSUM
Related Topics	N/A
Product Revision	Command and Firmware Release

NOTE: This command applies only to the ACR8020 and ACR9000 controller.

This command is used to program new firmware into the Sysflash1 area. Issue command **FIRMWARE CHECKSUM** and verify that the Sysflash2 Invalid/empty flag is cleared before this command is attempted. If the Sysflash2 code is valid and the **FIRMWARE UPGRADE** process fails, the bootloader can still load Sysflash2 code at power-up. If the Sysflash2 code is invalid and the **FIRMWARE UPGRADE** process fails, the controller needs to be sent back to the factory to be re-programmed.

ACR8020

The firmware upgrade procedure is as follows:

1. At SYS prompt type **HALT ALL** to halt all programs and plc programs.
2. Make sure all machines controlled by acr8020 are shut down.
3. At the prompt type **FLASH RES** to clear memory to factory default settings.
4. At the prompt "This will clear/reset everything on the controller. Continue (y/n)? ", answer y.
5. At the prompt "All programs and user settings will be lost. Continue (y/n)?", answer y.
6. Type **FIRMWARE CHECKSUM** and verify flag 5658 is cleared and flag 5660 is set, Otherwise type **FIRMWARE BACKUP** command to backup firmware first.
7. Type **FIRMWARE UPGRADE**.
8. At the prompt "Are you sure that you want to upgrade firmware(y/n)?", answer y.
9. Send ACR8020.dat as a text file. When the file has started loading, the display should read "`FIRMWARE UPGRADE START`"
10. Wait a few minutes until the display reads "`FIRMWARE UPGRADE COMPLETE`". Do not power down or perform any other functions until this operation is completed.
11. Depress the ACR8020 reset switch, SW2, and verify that the green watchdog LED is re-lit and that the communication port works.
12. Type **VER** and verify that the correct firmware version is running.

13. Type **FIRMWARE CHECKSUM** and verify that Sysflash1 checksum matches the provided checksum and flag 5657 is set.
14. Type **FIRMWARE BACKUP** to backup firmware. Do not power down or perform any other functions until this operation is completed.
15. Type **FIRMWARE CHECKSUM** and verify that Sysflash1 checksum and Sysflash1 checksum are identical, flag 5658 is cleared and flag 5660 is set.

ACR9000

The firmware upgrade procedure is as follows:

1. Open a terminal emulator (such as HyperTerminal) and set to 9600, 8 data, 1 stop, no parity, and no flow control. Connect to the ACR9000's com1 port.
2. Reset the ACR9000 while holding down the Ctrl and b keys. You should see and OSP> prompt.
3. Type baud –115200 and press enter. Change the terminal settings to match by selecting Call->disconnect. Then select File->Properties and click the configure button. Change the baud to 115200. Click OK. Click OK again. Type Enter a few times in the window until you see more OSP> prompts.
4. Right click and select Send File. Browse for the OS file you want to send. Copy the file path. Click Cancel.
5. Type osloader –r into the terminal. Right click and select Send File. Select 1K Xmodem for the protocol. Click into the filename textbox and paste the filename from above. This must be done quickly before the osloader times out. Click Send. The osloader should begin the download. This will take several minutes to download and save the file.
6. Change Hyperterminal's settings back to 9600 baud. Reset the ACR9000. Wait a few seconds and repeatedly hit Enter. You should see the SYS> prompt.
7. Close Hyperterminal and open ACR-View. The Operating System is now updated.

FLASH User Program Storage

Format	FLASH *command*
Group	Nonvolatile
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	N/A
See Also	PROM, ESAVE, ELOAD, ERASE
Related Topics	N/A
Product Revision	Command and Firmware Release

This command is used with a second command to manipulate an image of the user memory in the flash onboard the ACR1200, ACR1500, ACR1505, ACR2000, ACR8010, ACR8020, and ACR9000.

Caution — If data is present in the flash memory when you issue a FLASH SAVE or FLASH IMAGE command, the controller automatically issues a FLASH ERASE command. All data previously stored in the flash is then overwritten.

NOTE: ACR2000 only—For expanded user memory (512K) on the ACR2000 controller, flash storage is limited to 384K of program, PLC and user variable information.

NOTE: When using battery backup for RAM, do not use the FLASH commands. The FLASH commands write to the flash memory. On cycling power, the controller restores the flash memory to RAM, wiping out the data otherwise retained by battery backup.

The following is a list of valid flash command combinations:

FLASH ERASE: Erase user image from flash.

FLASH LOAD: Load user image from flash.

FLASH IMAGE: Save user variable and program image to flash (Version 1.17.07 & Up).

FLASH RES: Reset controller to factory settings.

FLASH SAVE: Save user program image to flash.

The power-up flash load can be bypassed, using the Flash Bypass Mode (Version 1.17.07 & Up). The Flash Bypass Mode is implemented by setting the ACR1200, ACR1500, ACR1505, ACR2000, ACR8010, ACR8020 Card Address Switch (SW1). Refer to the appropriate Hardware Manual for details.

When the Flash Bypass Mode is implemented the user programs, PLC's, and user variables (if a **FLASH IMAGE** command was used), are not loaded at power-up or reset. In addition the **PBOOT** commands in battery backup memory are disabled.

Example 1

The following example stores the user program image into flash, leaving the user variable in battery backup memory.

```
FLASH SAVE
```

Example 2

The following example provides a good example of saving all parameters and programs. Perform this only when all commanded motion is stopped.

```
HALT ALL
FLASH IMAGE
ESAVE ALL
```

FLASH ERASE User Program Storage

Format	FLASH ERASE
Group	Nonvolatile
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	PROM, ESAVE, ELOAD, ERASE
Related Topics	N/A
Product Revision	Command and Firmware Release

This command erases the user image from flash.

Example

The following example erases the user program image from flash.

```
FLASH ERASE
```

FLASH LOAD User Program Storage

Format	FLASH LOAD
Group	Nonvolatile
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	PROM, ESAVE, ELOAD, ERASE
Related Topics	N/A
Product Revision	Command and Firmware Release

This command loads the user image from flash.

Example

The following example loads the user program image from flash.

```
FLASH LOAD
```

FLASH IMAGE User Program Storage

Format	FLASH IMAGE
Group	Nonvolatile
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	PROM, ESAVE, ELOAD, ERASE
Related Topics	N/A
Product Revision	Command and Firmware Release

This command stores an image of the user programs and PLC programs, as well as the User Global Variables, in the flash onboard the ACR1200, ACR1500, ACR1505, ACR2000, ACR8010, ACR8020, and ACR9000.

If an image is detected in the flash on power-up, the controller loads the user programs, PLC's, and user variables from flash instead of relying on the battery backup memory.

NOTE: After issuing a **FLASH IMAGE** command, it can take up to 30 seconds to complete the storage.

Example

The following example provides a good example of saving all parameters and programs. Perform this only when all commanded motion is stopped.

```
HALT ALL
FLASH IMAGE
ESAVE ALL
```

FLASH RES User Program Storage

Format	FLASH RES
Group	Nonvolatile
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	PROM, ESAVE, ELOAD, ERASE
Related Topics	N/A
Product Revision	Command and Firmware Release

This command returns the controller to factory settings. It performs the following actions: It clears all memory dimensioned for programs, variable, arrays, and data logging; it resets the communication stream dimensions, erases all system parameters and user images from flash memory; finally, it resets the hardware configuration and cycles power to the controller.

When you send the **FLASH RES** command, the controller sends two separate warning messages before initiating the reset.

NOTE: Once the flash reset is completed, you have to upload your programs to the controller.

Example

The following example returns the controller to factory settings.

```
FLASH RES
```

FLASH SAVE User Program Storage

Format	FLASH SAVE
Group	Nonvolatile
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	PROM, ESAVE, ELOAD, ERASE
Related Topics	N/A
Product Revision	Command and Firmware Release

This command stores an image of the user programs and PLC programs in the flash onboard the ACR1200, ACR1500, ACR1505, ACR2000, ACR8010, ACR8020, and ACR9000. If the image is detected in the flash on power-up, the card will load user programs and PLC's from flash instead of relying on the battery backup memory. User variables reside in battery backup memory and are not affected by the program transfer.

Example

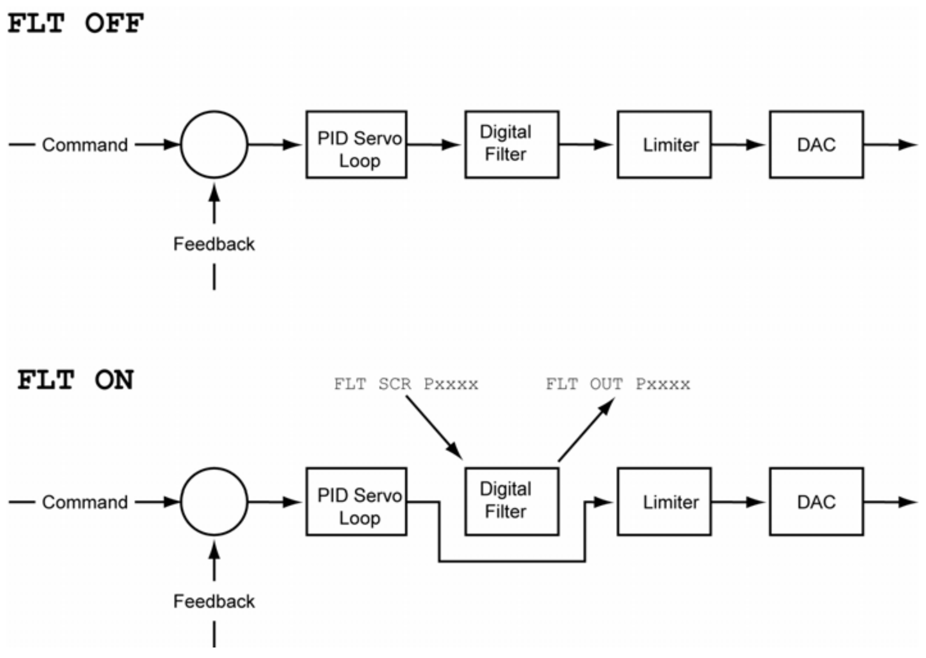
The following example stores the user program image into flash, leaving the user variable in battery backup memory:

```
FLASH SAVE
```

FLT Set Input and Output of the Digital Filter

Format	FLT *filter command* {*parameter*}
Group	Servo Control
Units	N/A
Data Type	N/A
Default	OFF
Prompt Level	SYS, PROGx, PLCx
See Also	FLT OFF, FLT ON, FLT OUT, FLT SRC, LOPASS, NOTCH
Related Topics	N/A
Product Revision	Command and Firmware Release

This command is used along with a second command to define the input and output of the digital filter in the servo loop. By using this command one can move the servo loop digital filter between any two P parameters. This gives the flexibility of filtering any P parameter on the controller. There are eight filters available (0-7).



NOTE: This filter is still updated in the servo loop block. The sequence in which the input, output and digital filter will update should be carefully sought, so that nothing is overwritten.

FLT OFF Put Back the Filter to its Default Location

Format	FLT *filter* OFF
Group	Servo Control
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	FLT ON, FLT OUT, FLT SRC, LOPASS, NOTCH
Related Topics	N/A
Product Revision	Command and Firmware Release

This command will put back the filter to its default location.

Example

```
FLT 0 OFF
```

FLT ON Turn On the Filter at the New Location

Format	FLT *filter* ON
Group	Servo Control
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	FLT OFF, FLT OUT, FLT SRC, LOPASS, NOTCH
Related Topics	N/A
Product Revision	Command and Firmware Release

FLT ON command will move the filter between **FLT SRC** and **FLT OUT** and start updating every servo loop. It will return an error if the source and output of the filter has not already been assigned.

Example

```
FLT 0 ON
```

FLT OUT Set Output of the Digital Filter

Format	FLT *filter* OUT {*parameter*}
Group	Servo Control
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	FLT OFF, FLT ON, FLT SRC, LOPASS, NOTCH
Related Topics	N/A
Product Revision	Command and Firmware Release

This command is used to set the output of the digital filter to a specific parameter.

Example

The output of digital filter 0 goes to vector velocity of master 1.

```
FLT 0 OUT P8449
```

FLT SRC Set Input Source of the Digital Filter

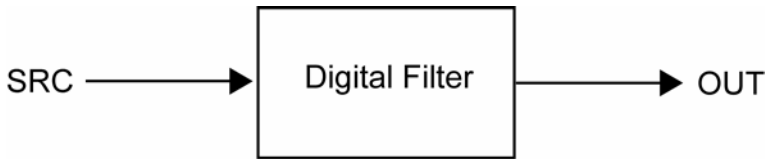
Format	FLT *filter* SRC {*parameter*}
Group	Servo Control
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	FLT OFF, FLT ON, FLT OUT, LOPASS, NOTCH
Related Topics	N/A
Product Revision	Command and Firmware Release

This command is used to set the input of the digital filter to a specific parameter.

Example

Current position of the axis 0 becomes the input of the digital filter 0.

```
FLT 0 SRC P12280
```



FLZ Relative Program Path Shift

Format	FLZ {*axis* {*shift*}} {*axis* {*shift*}} ...
Group	Transformation
Units	N/A
Data Type	FP32
Default	N/A
Prompt Level	PROGx
See Also	AXIS, OFFSET, ROTATE, SCALE
Related Topics	N/A
Product Revision	Command and Firmware Release

This command will cause the programmed path to be shifted. The amount of the path shift is defined by the "shift" relative to the current location. The program will think that the axis is currently at the location specified by the shift. If the shift for an axis is not specified, the offset will be cleared and any shift will be canceled.

Example

```
FLZ X10000 Y20000
```

FOR/TO/STEP/NEXT Relative Program Path Shift

Format	**FOR** countvalue = startvalue **TO** endvalue **STEP** stepvalue **NEXT**
Group	Program Flow
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	BIT, DEFINE, DIM, IF/THEN, IF/ELSE IF/ELSE/ENDIF
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

The **FOR/TO/STEP/NEXT** command executes a programmable loop based on three values in the control section. The loop is initialized by *startvalue*. Then the value the loop is incrementing towards, *endvalue*, is defined by the value after **TO**. The size of the increments used is defined by *stepvalue*, the value after the **STEP** command. Using the **NEXT** command encloses the loop.

Arguments

countvalue

Variable that supports integer data type. Aliases that refer to variables can be used.

startvalue

Must be an integer. Aliases can be used if referring to a constant. Variables are not accepted and will result in erroneous execution of the loop.

endvalue

Can be in integer value or a variable that supports integer data types. Aliases that refer to variables can be used. Expressions can be used if the resulting value is an integer. Variables, aliases, and expressions must be enclosed in parentheses.

stepvalue

Must be a positive integer. Aliases can be used if referring to a constant. Variables are not accepted and will result in erroneous execution of the loop. Negative values will also result in incorrect operation.

NOTE: You can use the **BREAK** command to interrupt the **FOR** loop when certain conditions are met.

NOTE: For **FOR/TO/STEP/NEXT** to function correctly, you must set the **DIM DEF** to a nonzero number.

Examples

For these examples, assume the following have been set using the ACR-View Configuration Wizard:

```
SYS
DIM PROG0(20000)
DIM P(4096)
DIM DEF(200)
```

Example 1

```
PROG0
PROGRAM
DIM LV(1)
FOR LV0 = 0 TO 9 STEP 1
      PRINT "LOOP # ";LV0
      NEXT
ENDP
```

Example 2

```
PROGRAM
FOR P1 = 1 TO 10 STEP 1
      PRINT "LOOP # "; P1
      NEXT
ENDP
```

Example 3

```
PROGRAM
DIM LV(2)
LV1=25
FOR LV0 = 0 TO (LV1) STEP 5
      PRINT "LOOP # "; LV0
      NEXT
ENDP

P00>LRUN
LOOP # 0
LOOP # 5
LOOP # 10
LOOP # 15
LOOP # 20
LOOP # 25
```

Example 4

```
#DEFINE lvIndex LV0
#DEFINE lvMAX LV1

PROGRAM
DIM LV(2)
lvMAX=10
FOR lvIndex = 0 TO (lvMAX-1) STEP 1
      PRINT "LOOP # "; lvIndex
      NEXT
ENDP

P00>LRUN
LOOP # 0
LOOP # 1
LOOP # 2
LOOP # 3
LOOP # 4
LOOP # 5
LOOP # 6
LOOP # 7
LOOP # 8
LOOP # 9
```

Example 5

```
#DEFINE counter LV2
#DEFINE STOP BIT128

PROGRAM
DIM LV(3)

FOR counter =0 TO 100 STEP 2
      PRINT "Counting "
      PRINT "Counter = ", counter
      IF (STOP)
            PRINT " Stop Counting"
            BREAK
            ENDIF
      PRINT " DWELL FOR 2 SEC"
      DWL 2
      NEXT
ENDP
```

Example 6

The following example counts to 25, by increments of 1. It uses a loop set by the **FOR/TO/STEP/NEXT** command, and uses a 1 second dwell (`DWL 1`) between increments. It stops counting when the STOPBIT bit is active, and pauses counting when the PAUSEBIT bit is active. Once paused, the program will resume counting upon activation of the RESUMEBIT.

The program uses both **GOTO** and **GOSUB** commands. The PAUSECOUNT routine uses a **GOSUB** because the program continues counting when the RESUMEBIT is set; therefore, it must remember where the jump originated. The STOPCOUNT and FINISHCOUNT functions use a **GOTO** command because they jump to the end of the program and do not need to return to the jump point.

```
#DEFINE COUNTER LV2
#DEFINE STOPBIT BIT131
#DEFINE PAUSEBIT BIT130
#DEFINE RESUMEBIT BIT129

PROGRAM
DIM LV(10)
FOR COUNTER = 1 TO 25 STEP 1
       PRINT "Counter = ";counter
       DWL 1

       IF (STOPBIT)
              GOTO STOPCOUNT
              ENDIF

       IF (PAUSEBIT)
              GOSUB PAUSECOUNT
              ENDIF

       NEXT
GOTO FINISHCOUNT

_STOPCOUNT
```

```
PRINT "Counting Stopped"
GOTO DONE

_PAUSECOUNT
PRINT "Counting Paused"
INH RESUMEBIT
RETURN

_FINISHCOUNT
PRINT "Counting Complete"

_DONE
ENDP
```

FOV Set Feedrate Override

Format	FOV {*rate*}
Group	Velocity Profile
Units	Multiplier of velocity
Data Type	FP32
Default	1
Prompt Level	PROGx
Report Back	Yes **To view, type FOV**
See Also	MASTER, ROV, VEL
Related Topics	Master Parameters (0-7) (8-15)
Product Revision	Command and Firmware Release

Description

Scales the velocity for the current coordinated motion profile.

Arguments

rate

Optional. Sets the percentage change to the **VEL** rate for the master profile. The rate is a multiplier, and only affects the coordinated motion profiler.

Remarks

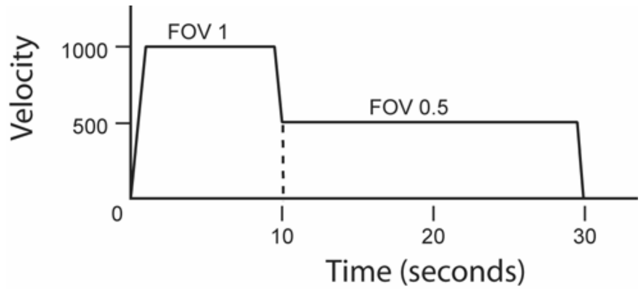
When you send **FOV**, the effect is immediate. The new velocity, as changed by **FOV**, remains in effect until you send another **FOV**. If a feed move is in progress, the master uses the current **ACC** or **DEC** rates to ramp to the new velocity.

The **FOV** command affects only the coordinated motion profiler, and does not affect the Jog, Gear, or Cam profilers.

NOTE: The **FOV** and **ROV** commands behave similarly. While the effect of **ROV** is the same as **FOV**, it has two bits controlling when it becomes active. Furthermore, when **ROV** is enabled **FOV** is disabled.

Example

The following example moves at velocity for ten seconds, then reduces velocity to 50 percent for the remainder of the move:



```
FOV 1      : REM ensures is set to default
DEC 1000 ACC 1000 STP 1000 VEL 1000
X/20000     : REM incremental move
DWL 10     : REM dwell for ten seconds
FOV 0.5     : REM scales velocity to 50 percent
```

FSTAT Fast Status Setup

Format 1:	FSTAT {*command* {*data*}}
Format 2:	FSTAT index1 {*command*}
Format 3:	FSTAT index1 (*code*, *index*)
Group	Global Objects
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS
See Also	PERIOD
Related Topics	FSTAT Flags, FSTAT Setup Parameters, Miscellaneous Parameters
Product Revision	Command and Firmware Release

NOTE: This command applies only to the ACR1505, ACR8020 and ACR900 controllers.

This command allows the system parameters being update to the dual port memory at the servo interrupt rate. The controller updates the dual port fast status periodically at the end of the servo loop update portion. The dual port fast status update frequency is set by the **FSTAT PERIOD** command. An interlocking mechanism is provided to prevent data fetching from the PC side while the controller is in the middle of dual port fast status update operation.

The following is a list of valid **FSTAT** command combinations:

Format 1

FSTAT: Display the setting of **FSTAT**.

FSTAT CLEAR: Clear the setting of all **FSTAT**.

FSTAT OFF: Disable dual port fast status update.

FSTAT ON: Enable dual port fast status update.

FSTAT PERIOD *period*: Set the dual port fast status update frequency.

Format 2

FSTAT *index1*: Display the setting of **FSTAT** *index1*.

Format 3

FSTAT *index1***(***code***,** *index***)**: Setup **FSTAT** *index1*.

Up to 10 groups of system parameters can be updated to the dual port memory simultaneously. Each group contains 8 parameters. Please refer to the ACR Programmer’s Guide section on Binary Data Packets for the selection of Group and Index.

Example

```
FSTAT CLEAR    : REM Clear FSTAT
FSTAT PERIOD 2    : REM Update FSTAT every other servo interrupt
FSTAT0(27,22)    : REM DPCB Status
FSTAT1(16,0)    : REM General Flags
FSTAT2(24,0)    : REM Encoder Position
FSTAT3(32,0)    : REM Master Distance into Move
FSTAT4(48,3)    : REM Axis Following Error
FSTAT5(32,1)    : REM Master Vector Velocity
FSTAT6(48,6)    : REM Primary Set Point
FSTAT7(32,9)    : REM FOV for Masters
FSTAT8(16,4)    : REM Program flags
FSTAT9(16,5)    : REM Program flags
FSTAT ON    : REM Turn on FSTAT
FSTAT    : REM Display the above setting
```

FSTAT CLEAR Clear Fast Status Settings

Format	FSTAT CLEAR
Group	Global Objects
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS
See Also	PERIOD
Related Topics	N/A
Product Revision	Command and Firmware Release

NOTE: This command applies only to the ACR1505, ACR8020 and ACR9000 controllers.

This command clears all fast status settings.

Example

```
FSTAT CLEAR    : REM Clear FSTAT
FSTAT PERIOD 2    : REM Update FSTAT every other servo interrupt
FSTAT0(27,22)    : REM DPCB Status
FSTAT1(16,0)    : REM General Flags
FSTAT2(24,0)    : REM Encoder Position
FSTAT3(32,0)    : REM Master Distance into Move
FSTAT4(48,3)    : REM Axis Following Error
FSTAT5(32,1)    : REM Master Vector Velocity
FSTAT6(48,6)    : REM Primary Set Point
FSTAT7(32,9)    : REM FOV for Masters
FSTAT8(16,4)    : REM Program flags
FSTAT9(16,5)    : REM Program flags
FSTAT ON    : REM Turn on FSTAT
FSTAT    : REM Display the above setting
```

FSTAT OFF Disable Fast Status Settings

Format	FSTAT OFF
Group	Global Objects
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS
See Also	PERIOD
Related Topics	N/A
Product Revision	Command and Firmware Release

NOTE: This command applies only to the ACR1505, ACR8020 and ACR900 controllers.

This command disables fast status.

FSTAT ON Enable Fast Status

Format	FSTAT ON
Group	Global Objects
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS
See Also	PERIOD
Related Topics	N/A
Product Revision	Command and Firmware Release

NOTE: This command applies only to the ACR1505, ACR8020 and ACR900 controllers.

This command enables fast status.

Example

```
FSTAT CLEAR    : REM Clear FSTAT
FSTAT PERIOD 2    : REM Update FSTAT every other servo interrupt
FSTAT0(27,22)    : REM DPCB Status
FSTAT1(16,0)    : REM General Flags
FSTAT2(24,0)    : REM Encoder Position
FSTAT3(32,0)    : REM Master Distance into Move
FSTAT4(48,3)    : REM Axis Following Error
FSTAT5(32,1)    : REM Master Vector Velocity
FSTAT6(48,6)    : REM Primary Set Point
FSTAT7(32,9)    : REM FOV for Masters
FSTAT8(16,4)    : REM Program flags
FSTAT9(16,5)    : REM Program flags
FSTAT ON    : REM Turn on FSTAT
FSTAT    : REM Display the above setting
```

FSTAT PERIOD Set Fast Status Update Frequency

Format	FSTAT PERIOD
Group	Global Objects
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS
See Also	PERIOD
Related Topics	N/A
Product Revision	Command and Firmware Release

NOTE: This command applies only to the ACR1505, ACR8020 and ACR900 controllers.

This command sets the dual-port fast-status update frequency.

Example

```
FSTAT CLEAR    : REM Clear FSTAT
FSTAT PERIOD 2    : REM Update FSTAT every other servo interrupt
FSTAT0(27,22)    : REM DPCB Status
FSTAT1(16,0)    : REM General Flags
FSTAT2(24,0)    : REM Encoder Position
FSTAT3(32,0)    : REM Master Distance into Move
FSTAT4(48,3)    : REM Axis Following Error
FSTAT5(32,1)    : REM Master Vector Velocity
FSTAT6(48,6)    : REM Primary Set Point
FSTAT7(32,9)    : REM FOV for Masters
FSTAT8(16,4)    : REM Program flags
FSTAT9(16,5)    : REM Program flags
FSTAT ON    : REM Turn on FSTAT
FSTAT    : REM Display the above setting
```

FVEL Set Final Velocity

Format	FVEL {*rate*}
Group	Velocity Profile
Units	units/second based on PPU
Data Type	FP32
Default	0
Prompt Level	PROGx
See Also	ACC, DEC, MASTER, IVEL, STP, VEL
Related Topics	N/A
Product Revision	Command and Firmware Release

This command sets the final velocity value for a master move profile. Final velocity is used as a target velocity when the **STP** ramp is active. The value is used to slow down, but not stop, between moves.

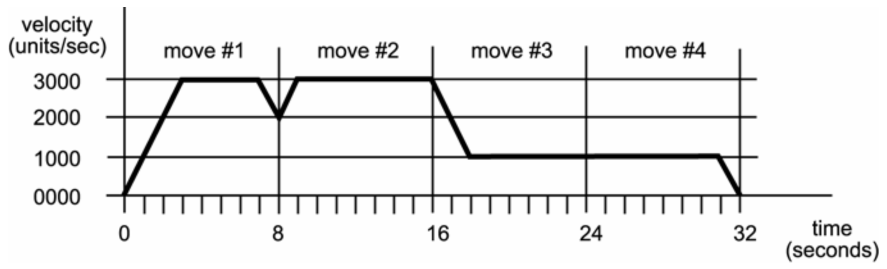
A move will not ramp up to this value, it will only ramp down. The final velocity is only used when **STP** is non-zero and the current velocity is greater than the final velocity.

Issuing an **FVEL** command without an argument will display the current setting. The default final velocity is zero. Regardless of the setting, the master bits "FVEL Zero Pending" and "FVEL Zero Active" can be used to temporarily override the final velocity to zero. An error will be returned if no master is attached.

Example

The following example generates a path using different combinations of velocity, final velocity, and stop ramps. Note that the velocity profile between moves 3 and 4 does not ramp down even though **STP** is set to 1000. This is because the final velocity of 2000 is greater than the current velocity at that point in the profile.

The following illustrates final velocity:



```
ACC1000 DEC1000
VEL3000 FVEL2000 STP1000 X/19000     : REM MOVE NUMBER 1
VEL3000 FVEL2000 STP0 X/23500     : REM MOVE NUMBER 2
VEL1000 FVEL2000 STP1000 X/10000     : REM MOVE NUMBER 3
VEL1000 FVEL0 STP1000 X/7500     : REM MOVE NUMBER 4
```

GEAR Electronic Gearing

Format	GEAR *command axis* {*data*} {*axis* {*data*}} ...
Group	Setpoint Control
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	AXIS, BKL, BSC, CAM, HDW, INTCAP, JOG
Related Topics	Axis Flags (0-7)(8-15),
Product Revision	Command and Firmware Release

Description

In electronic gearing, pulses are fed from a selected source into the gear offset of a slave axis. These pulses are scaled by a ratio that is equivalent to a gear ratio on a mechanical system. The rate at which the ratio changes is controlled by a ramping mechanism similar to a clutch or a variable speed gearbox.

Remarks

To view the current setting for a geared axis, send **GEAR {*axis*}**

Issuing the **GEAR** command displays the current setting of a gear, and can be used even if the gear is currently active.

The following is a list of valid electronic gearing command combinations:

GEAR ACC: Set gearing acceleration.

GEAR CLEAR: Clear electronic gearing settings.

GEAR DEC: Set gearing deceleration.

GEAR MAX: Set maximum gear offset limit.

GEAR MIN: Set minimum gear offset limit.

GEAR OFF: Turn electronic gearing off.

GEAR OFF TRG: Disable gear on external trigger.

GEAR OFF TRGP: Gear off by external trigger.

GEAR ON: Turn electronic gearing on.

GEAR ON IHPOS: Enable gear on source encoder position.

GEAR ON TRG: Enable gear on external trigger.

GEAR ON TRGP: enable gear on external trigger.

GEAR PPU: Scale electronic gearing input.

GEAR RATIO: Set electronic gearing ratio.

GEAR RES: Reset or preload gearing output.

GEAR SRC: Set electronic gearing source.

Gear or Handwheel?

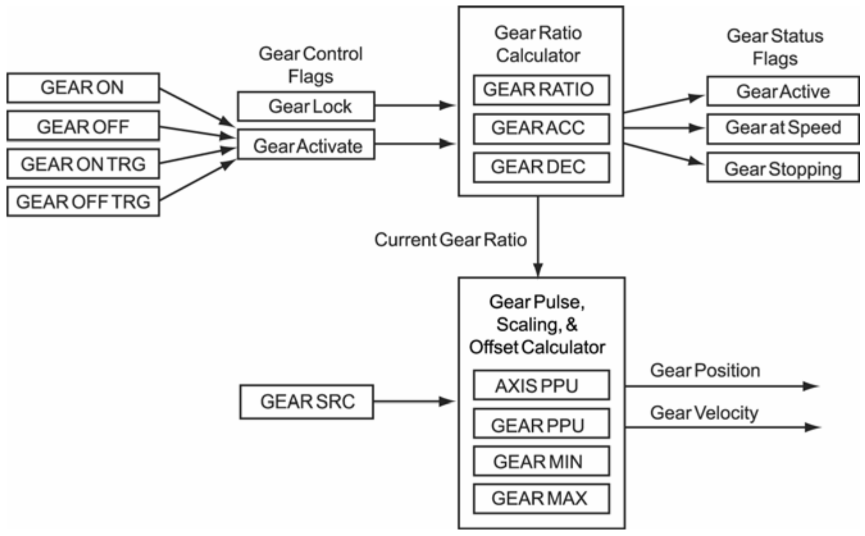
You can use the **GEAR** sub commands with the handwheel command (**HDW**) For example, **HDW ACC**, **HDW MAX** or **HDW RATIO**.

In the controller, handwheels and gears use the same internal mechanism. There is no difference between the two command sets. The following are equivalent:

```
P00> GEAR ON X
P00> HDW ON X
```

Block Diagram

The following illustrates the operation of electronic gearing:



For more information, see the Position Velocity Servo Loop diagram on page 40.

Examples

Example 1

The following example demonstrates the use of the **GEAR ACC**, **GEAR DEC**, **GEAR MAX**, **GEAR MIN**, **GEAR PPU**, **GEAR RATIO**, **GEAR SRC**, and **GEAR ON TRG** commands. The program assumes that axis 2 is attached to master 0, axis 0 is geared to encoder 8, and axis 1 is geared to encoder 0.

```
gear off x
gear off y
drive off x y

res x y
gear res x
gear res y
rem clear enc8
p6272 = 0

rem acceleration is in ppu. if set to zero, infinite acceleration
gear acc x0 y10
rem deceleration is in ppu. if set to zero, infinite deceleration
gear dec x0 y10
rem positive maximum gear position, if set 0 gear distance is infinite.
gear max x0 y8192
rem negative maximum gear position, if set 0 gear distance is infinite.
gear min x0 y-8192
```

```
rem set the gear pulses per unit (ppu)to the master encoder resolution
gear ppu x4000 y8192
rem gear slave ratio slave/master
gear ratio x1 y(-1)
rem set slave y to follow master encoder signal encoder 0
rem set slave x to follow master encoder signal encoder 8
gear src x enc8 y enc0

rem enable drive on axis x y
drive on x y
rem turn on gearing based on trigger. "x"
rem "gear on x y" would also work
rem axis0 "trg( rising primary external, register)"

gear on x trg(2,0)
gear on y trg(2,0)

_loop
rem if bit26 is true, jog incremental distance on axis y(gear slip/ppu)
if (bit26) then jog inc y(p12625/(p12631))
goto loop

rem turned on after drive is enabled, and disabled
rem before drive can be disabled.
```

Example 2

The following example will cause X axis to follow encoder 1 at a 1:4 ratio. Note that the **PPU** values equate to 0.25 inches per revolution (IPR). The slave axis will move 0.25 inches for every revolution of the electronic gearing source encoder.

```
PPU X10000     : REM Slave is 10000 pulses per inch
GEAR SRC X1     : REM Tie slave's gearbox to ENC1
GEAR PPU X1000     : REM Master is 1000 pulses per rev
GEAR RATIO X.25     : REM Set gear ratio at 1:4 (0.25 IPR)
GEAR ON X    : REM Turn electronic gearing on
```

Example 3

The following example uses the **GEAR ON TRG** and **GEAR OFF TRG** commands to control when the gear is enabled.

NOTE: GEAR ON TRG and **GEAR OFF TRG** are available in Firmware Version 1.18.04 & Up.

```
ATTACH MASTER0
ATTACH SLAVE0 AXIS0 "X"
ATTACH SLAVE1 AXIS1 "Y"

GEAR ACC X10000
GEAR DEC Y10000
GEAR SRC Y0
GEAR RATIO Y1
X/ 200000
GEAR ON Y TRG(2, 0) OFFSET 3000
REM Mode 2, Rising Primary external.
REM Capture Register 0, gear source is ENC0.
REM Offset is positive, X-axis is moving in positive direction.
INH 2344
REM Wait, capture register is shared by GEAR TRG ON and GEAR TRG OFF.
GEAR OFF Y TRG(2,0) OFFSET 6500
REM The gear will turn off 6500 pulses after the trigger is received.
INH 2348
```

Example 4

The example below shows the list:

```
P00>GEAR Y
GEAR ACC Y0
GEAR DEC Y0
GEAR MAX Y100
GEAR MIN Y-100
GEAR PPU Y1000
GEAR RATIO Y1
GEAR SRC Y ENC 3
GEAR ON Y
```

GEAR ACC Set Gearing Acceleration

Format	GEAR ACC *axis* {accel} {*axis* {accel}} ...
Group	Setpoint Control
Units	output units / input unit / second or output units / input unit / input unit
Data Type	FP32
Default	0
Prompt Level	PROGx
See Also	AXIS, BKL, BSC, CAM, GEAR, HDW, JOG
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Sets the rate at which the gear ratio will change when the target gear ratio is higher than the current ratio. This will occur both when gearing is turned on and when a higher gear ratio is set with the **GEAR RATIO** command.

Remarks

Setting gearing acceleration to 0.0 (default) or setting the "Gear Lock" flag (e.g., Bit Index 13 of the Primary Axis Flags) will cause an immediate lock. Issuing a **GEAR ACC** command to an axis without an argument will display the current setting for that axis.

NOTE: See Axis Parameter "Gear Slip".

NOTE: For more code examples, see the GEAR command.

By default, the **GEAR ACC** value is interpreted as *output units / input unit / second* (change in ratio per second). But if Bit Index 11 in secondary axis flags (Bit 2315 for Axis 0) is set, then the **GEAR ACC** value is interpreted as *output units / input unit / input unit* (change in ratio per input unit.) This results in a ratio ramp that always takes place over the same amount of source travel, regardless of the speed of the source. It also allows precise calculation of the output change during the ramp, regardless of the speed of the source. For example, suppose the ratio needs to ramp from zero to two over 0.5 units of source travel. Then the required **GEAR ACC** would be 4 (2 / 0.5 = 4). During the ramp, the average ratio is just exactly one, so the output change would be exactly 0.5 (1 x 0.5 = 0.5).

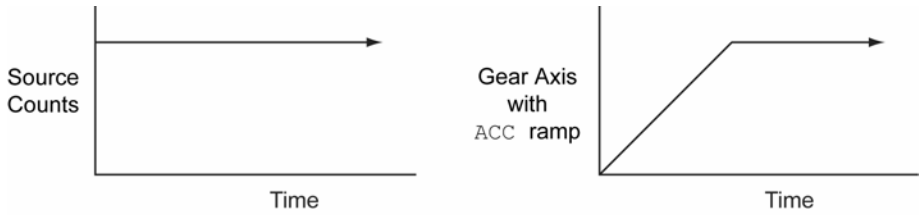
Example

The following sets the X axis gearing acceleration to 2.

```
GEAR ACC X2
```

GEAR ACC SLIP (Version 1.18.04)

When the gear acceleration or deceleration is not zero, then the geared axis will take some time to ramp up or down to the new ratio. During this time, the output effectively "slips" compared to what it would have been if the acceleration or deceleration were zero. The "Gear Slip" axis parameter stores the number of output pulses "slipped" before the geared axis is at the new ratio. This gives the flexibility to recover these missed counts by adding an incremental move (in counts) to the geared axis by the change of "Gear Slip" value. This value accumulates with each change in ratio, so to correct for the slip in a ratio ramp, the change in Gear Slip must be used.



GEAR CLEAR Clear Electronic Gearing Settings

Format	GEAR CLEAR {*axis*}
Group	Setpoint control
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	AXIS, GEAR
Related Topics	N/A
Product Revision	Command and Firmware Release

This command will clear the current setting of a gear. It will turn off the gear and then reset the gear variables to their initial default values.

NOTE: For more code examples, see the GEAR command.

Example

The following example illustrates how the **GEAR** settings are cleared on a controller.

```
P00>gear x
GEAR ACC X 100
GEAR DEC X 100
GEAR MAX X 250

GEAR MIN X0
GEAR PPU X1
GEAR RATIO X1
GEAR SRC X NONE
GEAR OFF X

P00>gear clear x
P00>gear x
GEAR ACC X0
GEAR DEC X0
GEAR MAX X0

GEAR MIN X0
GEAR PPU X1
GEAR RATIO X1
GEAR SRC X NONE
GEAR OFF X
```

GEAR DEC Set Gearing Deceleration

Format	GEAR DEC *axis* {*decel*} {*axis* {*decel*}} ...
Group	Setpoint Control
Units	output units / input unit / second or output units / input unit / input unit
Data Type	FP32
Default	0
Prompt Level	PROGx
See Also	AXIS, BKL, BSC, CAM, GEAR, HDW, JOG
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Sets the rate at which the gear ratio will change when the target gear ratio is lower than the current ratio. This will occur both when gearing is turned off and when a lower gear ratio is set with the **GEAR RATIO** command.

Remarks

Setting gearing deceleration to 0.0 (default) or setting the "Gear Lock" flag (e.g., Bit Index 13 of the Primary Axis Flags) will cause an immediate lock. Issuing a **GEAR DEC** command to an axis without an argument will display the current setting for that axis.

NOTE: For more code examples, see the GEAR command.

By default, the **GEAR DEC** value is interpreted as *output units / input unit / second* (change in ratio per second). But if Bit Index 11 in secondary axis flags (Bit 2315 for axis 0) is set, then the **GEAR DEC** value is interpreted as *output units / input unit / input unit* (change in ratio per input unit). This results in a ratio ramp that always takes place over the same amount of source travel, regardless of the speed of the source. It also allows precise calculation of the output change during the ramp, regardless of the speed of the source. For example, suppose the ratio needs to ramp from four to two over 0.5 units of source travel. Then the required **GEAR DEC** would be 2 / 0.5 = 4. During the ramp, the average ratio is just exactly three, so the output change would be exactly 3 x 0.5 = 1.5.

Example

The following example sets the X axis gearing deceleration to 5.0:

```
GEAR DEC X5
```

GEAR MAX Set Maximum Gear Offset Limit

Format	GEAR MAX *axis* {*value*} {*axis* {*value*}} ...
Group	Setpoint Control
Units	Counts
Data Type	Long
Default	0
Prompt Level	PROGx
See Also	AXIS, BKL, BSC, CAM, GEAR, HDW, JOG
Related Topics	Secondary Axis Flags (0-7) (8-15)
Product Revision	Command and Firmware Release

Description

This command sets the maximum gear offset limit for the given axis. The maximum gear offset is defined by the offset relative to the current location of the gear source.

Remarks

Issuing the **GEAR MAX** command to an axis without an argument displays the maximum limit for that axis. The default is 0.0 for all axes.

NOTE: For more code examples, see the GEAR command.

Example

The following example sets the offset maximum for the Y axis gear to +1000 counts:

```
GEAR MAX Y -1000
```

Secondary Axis Flag “Gear Max” (Version 1.18.06 Update 09)

When the gear max limit is hit, this flag is automatically set. It self clears when gear comes back within the max limit.

GEAR MIN Set Minimum Gear Offset Limit

Format	GEAR MIN *axis* {*value*} {*axis* {*value*}} ...
Group	Setpoint Control
Units	Counts
Data Type	Long
Default	0
Prompt Level	PROGx
See Also	AXIS, BKL, BSC, CAM, GEAR, HDW, JOG
Related Topics	Secondary Axis Flags (0-7) (8-15)
Product Revision	Command and Firmware Release

Description

This command sets the minimum gear offset limit for the given axis. The minimum gear offset is defined by the offset relative to the current location of the gear source.

Remarks

Issuing the **GEAR MIN** command to an axis without an argument displays the minimum limit for that axis. The default is 0.0 for all axes.

NOTE: For more code examples, see the GEAR command.

Example

The following example sets the offset minimum for the Y axis gear to -1000 counts:

```
GEAR MIN Y -1000
```

Secondary Axis Flag "Gear Min" (Version 1.18.06 Update 09)

When the gear min limit is hit, this flag is automatically set. It self clears when gear comes back within the min limit.

GEAR OFF Turn Electronic Gearing Off

Format	GEAR OFF *axis* {*offset*} {*axis* {*offset*}} ...
Group	Setpoint Control
Units	output units
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	N/A
Related Topics	Axis Flags (0-7)(8-15)
Product Revision	Command and Firmware Release

This command disables electronic gearing for an axis. If the optional *offset* argument is left out, it is ignored. Otherwise the offset is preloaded to the given value.

The difference between the old offset and the new offset will show up in the axis current position. This prevents the axis from jumping when the gear offset changes.

NOTE: For more code examples, see the **GEAR** command.

Example

The following example disables electronic gearing on the X and Y axis:

```
GEAR OFF X Y
```

GEAR OFF TRG Gear Off by External Source Trigger

Format	GEAR OFF {*axis*} TRG (*mode*, *capture_register*) {OFFSET {*value*}}
Group	Setpoint Control
Units	Offset= input units
Data Type	Offset= FP32
Default	N/A
Prompt Level	PROGx
See Also	AXIS, GEAR, INTCAP, SRC
Related Topics	Axis Parameters (0-7) (8-15)
Product Revision	Command and Firmware Release

NOTE: This command does not apply to the ACR8000 controller.

This command arms the **GEAR** to stop when an externally sourced trigger occurs. The latency error is 1 microsecond (400 nanoseconds for the ACR9000). The mode parameter and hardware capture register information for the **GEAR ON TRG** is the same as those used in the **INTCAP** command.

The offset is the number of pulses from the trigger point to where the gear will be turned off. It is stored in the axis parameter "Gear Trigger Off Offset". The offset should be a positive number if the gear source is moving in the positive direction, and vice versa. The default offset value is zero, which will immediately turn off the gear.

NOTE: For more code examples, see the GEAR command.

Example

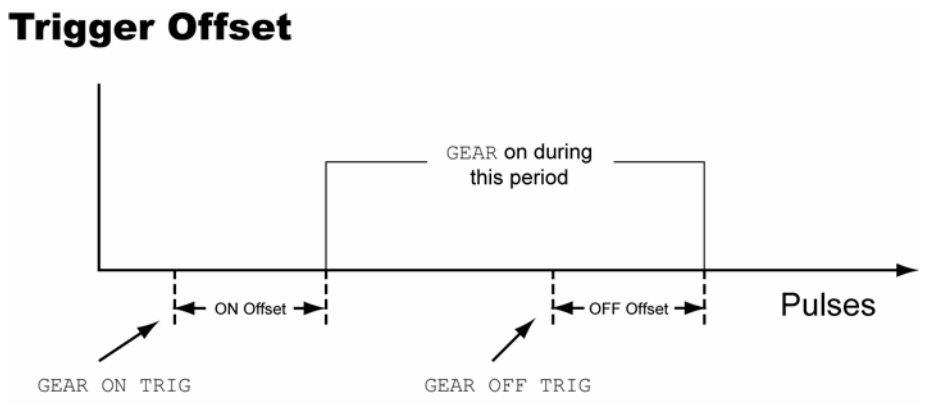
```
GEAR SRC Y ENC0
:
REM Mode = Primary Rising Marker; Capture Register = 1
GEAR OFF Y TRG(0,1)
```

GEAR OFF TRGP Gear Off by External Trigger

Format	GEAR OFF {*axis*} TRGP(*mode*, *capture_register*) {OFFSET {*value*}}
Group	Setpoint Control
Units	Offset= input units
Data Type	Offset= FP32
Default	N/A
Prompt Level	PROGx
See Also	AXIS, GEAR, INTCAP, SRC
Related Topics	N/A
Product Revision	Command and Firmware Release

NOTE: This command does not apply to the ACR8000 controller.

This command is same as the **GEAR OFF TRG**, except that gear can be triggered from any P parameter. In this case the capture register value is not used, since it is different from the gear source value. The resulting response is less precise than when triggered from the gear source. The worse case latency error could be up to one servo period.



NOTE: For more code examples, see the GEAR command.

Example

The following example sets the source input for axis Y, and sets the trigger mode and capture register.

```
GEAR SRC Y P12288
:
REM Mode = Primary Rising Marker; Capture Register = 1
GEAR OFF Y TRGP(0,1)
```

GEAR ON Turn Electronic Gearing On

Format	GEAR ON *axis* {*offset*} {*axis* {*offset*}} ...
Group	Setpoint Control
Units	output units
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	AXIS, BKL, BSC, CAM, GEAR, HDW, JOG
Related Topics	Axis Flags (0-7)(8-15)
Product Revision	Command and Firmware Release

This command enables electronic gearing for an axis. If you do not include the *offset* argument, it is ignored. Otherwise the offset is preloaded to the given value.

The offset is a multiplier for the PPU. The difference between the old offset and the new offset will show up in the axis current position. This prevents the axis from jumping when the gear offset changes.

NOTE: For more code examples, see the GEAR command.

Example

The following example turns on electronic gearing for axis X, Y and Z. The X axis gear offset is preloaded to 1000.

```
GEAR ON X1000 Y Z
```

GEAR ON IHPOS Enable Gear on Source Encoder Position

Format 1	GEAR ON {*axis*} + IHPOS {*setpoint*}
Group	Setpoint control
Units	Setpoint= input units
Data Type	Setpoint= FP32
Default	N/A
Prompt Level	PROGx
See Also	AXIS, GEAR, IHPOS
Related Topics	Secondary Axis Flags (0-7) (8-15)
Product Revision	Command and Firmware Release

This command arms the **GEAR** to begin when the gear source is equal to and going greater or less than the setpoint value. In the command syntax, use the following to indicate direction:

- Plus sign (+)—the source is greater than or equal to the setpoint value.
- Minus sign (-)—the source is less than or equal to the setpoint value.

This command arms the **GEAR** to begin when the gear source becomes equal to and going less then the setpoint value when using the –

The Gear Offset Register will be accurate to the setpoint value without any source encoder counts loss.

This command does not inhibit program operation. Once this command is issued, the program will continue to process commands. The **GEAR** will start when the setpoint conditions are met. The **GEAR ON IHPOS** command can be aborted with the **GEAR OFF** command.

In the Tertiary Axis Flags, you can use the “GEAR ON IHPOS Armed” flag to clear **GEAR ON IHPOS**.

NOTE: For more code examples, see the GEAR command.

Example

```
ATTACH MASTER0
ATTACH SLAVE0 AXIS0 "X"
ATTACH SLAVE1 AXIS1 "Y"
PPU X1 Y1
VEL 200
ACC 0 DEC 0 STP 0
GEAR RATIO Y 1

GEAR OFF Y
RES X Y
GEAR SRC Y P12288
LV1 = 1 : LV2 = 500
_LOOP1
IF (LV1 = 10) THEN GOTO LOOP2
X /1000
GEAR ON Y IHPOS + (LV2)
DWL 6
GEAR OFF Y
```

```
LV2 = (LV2 + 1000)
LV1 = (LV1 + 1)
GOTO LOOP1

_LOOP2
END
DIM LV5
```

NOTE: The **GEAR SRC** in this program is the Current Position Register of X.

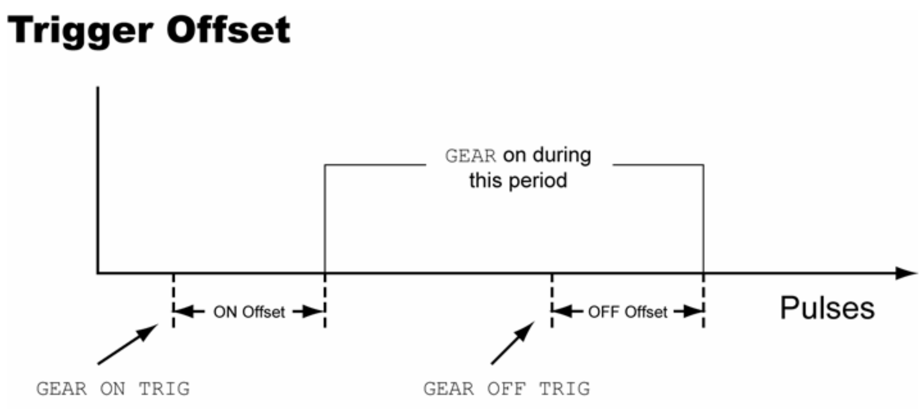
GEAR ON TRG Enable Gear On External Source Trigger

Format	GEAR ON {*axis* {*offset*}} TRG(*mode*, *capture_register*) {OFFSET {*value*}}
Group	Setpoint Control
Units	Offset= input units
Data Type	Offset= FP32
Default	N/A
Prompt Level	PROGx
See Also	GEAR, INTCAP, SRC
Related Topics	Axis Parameters (0-7) (8-15), Secondary Axis Flags (0-7) (8-15)
Product Revision	Command and Firmware Release

NOTE: This command does not apply to the ACR8000 controller.

This command arms the **GEAR** to begin when an external source trigger occurs. The latency error is 1 microsecond (400 nanoseconds for the ACR9000). The mode parameter and hardware capture register information for the **GEAR ON TRG** is the same as those used in the **INTCAP** command.

The offset is the number of pulses from the trigger point to where the gear will be turned on. It is stored in the axis parameter " Gear Trigger On Offset". The offset should be positive if the gear source is moving in the positive direction, and vice versa. The default offset value is zero, which will immediately turn on the gear.



NOTE: For more code examples, see the GEAR command.

Example

```
REM Gear Source is the actual position of axis0
GEAR SRC Y ENC0
REM ACR8010;
REM Mode = Primary Rising External (INP 24); Cap Register=0
GEAR ON Y TRG(2,0)
X/90
INH 809 : REM wait for capture complete.
REM Mode = Primary Rising Marker; Capture Register=1
GEAR OFF Y TRG(0,1)
```

GEAR ON TRGP Enable Gear On External Trigger

Format	GEAR ON {*axis* {*offset*}} TRGP(*mode*, *capture_register*) {OFFSET {*value*}}
Group	Setpoint Control
Units	Offset= input units
Data Type	Offset= FP32
Default	N/A
Prompt Level	PROGx
See Also	AXIS, GEAR, INTCAP, SRC
Related Topics	N/A
Product Revision	Command and Firmware Release

NOTE: This command does not apply to the ACR8000 controller.

This command is same as the **GEAR ON TRG**, except that gear can be triggered from any P parameter. In this case the capture register value is not used since it is different from the gear source value. The result is a less precise response than when triggered from the gear source. The worse case latency error could be up to one servo period.

NOTE: For more code examples, the GEAR command.

Example

```
REM Gear Source is the current position of axis0
GEAR SRC Y P12288
REM ACR8010;
REM Mode = Primary Rising External (INP 24); Cap Register = 0
GEAR ON Y TRGP(2,0)
X/90
INH 809 : REM wait for capture complete.
GEAR OFF Y
```

GEAR PPU Set Gearing Pulses Per Unit

Format	GEAR PPU *axis* {*ppu*} {*axis* {*ppu*}} ...
Group	Setpoint Control
Units	input pulses / input unit
Data Type	FP32
Default	1
Prompt Level	PROGx
See Also	AXIS, BKL, BSC, CAM, GEAR, HDW, JOG
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

This command establishes the relationship between the source encoder and the "input shaft" of the electronic gearbox.

Remarks

The **GEAR RATIO** command is responsible for setting the ratio between the input and output shafts. Issuing a **GEAR PPU** command to an axis without an argument will display the current setting for that axis. The default gearing pulses per unit is 1.0 for all axes.

The **GEAR PPU** is the divisor for the incoming **GEAR SRC** counts before the **GEAR RATIO** is applied.

Examples

For more code examples, see the GEAR command.

Example 1

Sets up the X axis (AXIS0) "input shaft" for 1000 pulses per unit:

```
GEAR PPU X1000
```

or

```
AXIS0 GEAR PPU 1000
```

Example 2

X Axis uses the Y Axis (AXIS1) as the gear source. Y axis has an 8000 line encoder and 5 mm/lead ballscrew. PPU for Y is 1600 counts per mm.

```
GEAR SRC X P12546 : REM source is actual position of Y Axis
GEAR PPU X1600 : REM gear ppu for X is the ppu of Y Axis
```

or

```
GEAR PPU X P12631 : REM Y Axis ppu is stored in P12631
```

GEAR RATIO Set Electronic Gearing Ratio

Format	GEAR RATIO *axis* {*ratio*} {*axis* {*ratio*}} ...
Group	Setpoint Control
Units	output units / input unit
Data Type	FP32
Default	1
Prompt Level	PROGx
See Also	AXIS, BKL, BSC, CAM, GEAR, HDW, JOG
Related Topics	N/A
Product Revision	Command and Firmware Release

This command sets the ratio between the "input shaft" and the "output shaft" of an electronic gearbox. The "speed" of the output shaft is equal to the speed of the input shaft multiplied by this ratio. Issuing a **GEAR RATIO** command to an axis without an argument will display the current setting for that axis. The default gearing pulses per unit is 1.0 for all axes.

NOTE: Use an actual ratio (for example 1/3) instead of a decimal approximation (for example .0333) to prevent rounding errors and drift.

NOTE: For more code examples, see the GEAR command.

Example

The following example sets up the Y axis gearbox for a 1:10 ratio:

```
GEAR RATIO Y(1/10)
```

GEAR RES Reset or Preload Gearing Output

Format	GEAR RES *axis* {*offset*} {*axis* {*offset*}} ...
Group	Setpoint Control
Units	output units
Data Type	FP32
Default	1
Prompt Level	PROGx
See Also	AXIS, BKL, BSC, CAM, GEAR, HDW, JOG
Related Topics	Axis Parameters (0-7) (8-15)
Product Revision	Command and Firmware Release

This command either clears or preloads the gear offset for the given axis. If the *offset* argument is left out, the gearing offset is set to zero. Otherwise the offset is preloaded to the given value.

The difference between the old offset and the new offset will show up in the axis current position. This prevents the axis from jumping when the gear offset changes.

NOTE: For more code examples, see the GEAR command.

Example

The following example clears out the gear offset for the Z axis:

```
GEAR RES Z
```

GEAR SRC Set Electronic Gearing Source

Format	GEAR SRC *axis sourcedef* {*axis sourcedef*} ...
Group	Setpoint Control
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	AXIS, GEAR, SRC
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Specifies the source for the input of an electronic gearbox. See the SRC command for the definition of the *sourcedef* argument.

Examples

For more code examples, see the GEAR command.

Example 1

Gear source for X Axis is encoder 1.

```
GEAR SRC X1
```

Note that when using an encoder as the gear source, the following syntax is also valid:

```
GEAR SRC X ENC1
```

or

```
GEAR SRC X P6160    : REM parameter for Encoder 1
```

Example 2

In an EPL system (ACR9030 or ACR9040) the feedback devices for the drives are not populated to encoder objects, but rather to EPLD objects. Assume X is AXIS0/EPLD0 and Y is AXIS1/EPLD1:

```
GEAR SRC X EPLD1    : REM "EPLD1 POSITION" parameter
```

or

```
GEAR SRC X P12546    : REM actual position for AXIS1
```

Example 3

Commanded position values are often chosen as the gear source. Assume X is AXIS0 and Y is AXIS1:

```
GEAR SRC X P12551
REM X is geared to the secondary setpoint (total commanded position) of
REM Y axis
```

```
GEAR SRC X P12544
REM X is geared to the current position (coordinate move profiler) of
REM Y axis

GEAR SRC X P12553
REM X is geared to the jog offset of the Y axis
```

GOSUB Branch to a Subroutine

Format	GOSUB *line*
Group	Program Flow
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	GOTO, DIM, RETURN
Related Topics	N/A
Product Revision	Command and Firmware Release

This command causes an unconditional branch to a subroutine. Each subroutine call requires 4 bytes of free memory to store its return address. Each subroutine must be terminated with a **RETURN** command

Example 1

The following example illustrates the **GOSUB** routine in a non-motion program.

```
_LOOP1
IF (BIT1) THEN GOSUB LOOP2 : PRINT "Back from Loop2" : GOTO LOOP1
IF (BIT2) THEN GOSUB LOOP3 : PRINT "Back from Loop3" : GOTO LOOP1
GOTO LOOP1

_LOOP2
Print "Loop2"
RETURN

_LOOP3
Print "Loop3"
RETURN
```

Example 2

The following example illustrates the **GOSUB** routine in a motion program.

```
REM main program loop
_LOOP1
IF (BIT1) THEN GOSUB LOOP2
IF (BIT2) THEN GOSUB LOOP3
GOTO LOOP1

REM first subroutine
_LOOP2
X10000
INH -516 : REM inhibit program until not in motion
RETURN

REM second subroutine
_LOOP3
X0
INH -516 : REM inhibit program until not in motion
RETURN
```

GOTO Branch to a New Line Number

Format	GOTO *line*
Group	Program Flow
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	DIM, GOSUB, RETURN
Related Topics	N/A
Product Revision	Command and Firmware Release

The command causes an unconditional branch to occur without clearing the nesting of loops, **IF**s and **GOSUB**s.

Example

```
ACC10 DEC10 STP10 vel1
_LOOP1
SET 32
X/1
INH -768
CLR 32
DWL 2
GOTO LOOP1
```

HALT Halt an Executing Program

Format	HALT {PROG *number* | PLC *number* | ALL}
Group	Program Control
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	RUN, LRUN, LISTEN
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Stops the execution of a running program or PLC, and kills any master (coordinate) motion profile initiated by the program.

Arguments

PROG *number*	Specifies program number.
PLC *number*	Specifies PLC number.
ALL	Specifies all programs and all PLCs.

Remarks

The **HALT** command with arguments can be used at any prompt level. **HALT PROG** and **HALT PLC** stop the corresponding user or PLC program. **HALT ALL** stops all user and PLC programs.

The **HALT** command without an optional argument cannot be issued from within a program; see END.

NOTE: The **HALT** command does not stop jog, gear, or cam motion. To stop these types of motion use **JOG OFF**, **GEAR OFF**, and **CAM OFF** respectively.

Example

The following demonstrates different ways to stop program execution.

```
P00>HALT    : REM Stops execution of current program or PLC
P00>HALT PROG1    : REM Stop execution of program 1
P00>HALT All    : REM Stop execution of all PLCs and programs
P00>HALT PLC1    : REM Stop execution of PLC1
```

The following demonstrates different ways of using **HALT** within programs:

```
REM Correct Usage
PROG0
PROGRAM
_START
HALT PROG3    : REM Can be placed within a program
IF (BIT 24) THEN HALT PROG1    : REM Can be used with IF/THEN command
IF (BIT 25)
    HALT PLC0    : REM …or can be used with IF/ENDIF command
    ENDIF
```

```
IF (BIT 26)
    HALT PROG0    : REM Can be used to halt current program
    ENDIF
GOTO START
ENDP

REM Incorrect Usage
PROG0
PROGRAM
HALT    : REM Lack of argument within a program
       : REM generates a syntax error
ENDP
```

HDW Handwheel

Format	HDW *command* {*axis* {*data*}} {*axis* {*data*}} ...
Group	Setpoint Control
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	AXIS, BKL, BSC, CAM, GEAR, JOG
Related Topics	N/A
Product Revision	Command and Firmware Release

Handwheel is another name for the **GEAR** command used for electronic gearing. See description of GEAR command for details.

HELP Display Command List

Format	HELP
Group	Operating System
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	DIAG
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Displays the executive version and command set.

Remarks

The **HELP** command cannot be issued from within a program.

Example

```
HELP
```

HLBIT Assign Hardware Limit and Homing Inputs

		Product	Revision
Format	HLBIT {*axis* {input}}	ACR1505	n/a
Group	Axis Limits	ACR8020	n/a
Units	N/A	ACR9000	1.18.15
Data Type	N/A	ACR9030	all
Default	See Below	ACR9040	all
Prompt Level	PROGx	Aries CE	all
See Also	HLDEC, HLIM		
Related Topics	N/A		

Description

This command assigns the inputs used for positive and negative hardware limits, and the input used for homing. The factory default sets the lowest onboard inputs to the lowest axis—axis 0 uses inputs 0, 1,and 2; axis 1 uses inputs 3, 4, and 5; and so on.

Remarks

You can change the input assignments using the **HLBIT** command (no corresponding parameter exists).

The value you provide sets the input to use as the positive hardware limit. The controller automatically sets the next contiguous input for the negative hardware limit, and sets the next contiguous input for homing.

NOTE: There are no restrictions regarding how to assign hardware limits and homing inputs. However, you should exercise caution because it is possible to create imaginary limit and home inputs. For example, if the positive hardware limit is assigned to input 31, the negative hardware limit and homing inputs are not assigned. Instead, they become imaginary inputs with a value of zero. If setting **HLBIT31**, it is recommended to also set **HLIM1**.

The **HLBIT** mode can be saved using the **ESAVE** command.

Example

The following example assigns the positive and negative hardware limits and homing to inputs 3, 4, and 5 respectively.

```
HLBIT Y3
```

HLDEC Hardware Limit Deceleration

Format	HLDEC {*axis* {*units*}}
Group	Axis Limits
Units	Units
Data Type	FP32
Default	0
Prompt Level	PROGx
See Also	HLBIT, HLIM
Related Topics	N/A
Product Revision	Command and Firmware Release

This command sets the rate of deceleration used after hitting a hardware end-of-travel limit. Motion stops at the **HLDEC** rate under the following circumstances:

- Hardware end-of-travel limit is reached
- Kill command (CTRL+X or CTRL+Z) is sent
- Kill All Motion Request is set (bit 19, Quaternary Axis flags)

When the controller receives end-of-travel limit input or a Kill command and the Disable Drive On Kill bit is set (bit 23, Quaternary Axis flags), the drive shuts down immediately—this allows the motor/load to freewheel to a stop without a controlled deceleration.

Example

The following example set the deceleration for axis X to 50,000.

```
HLDEC X50000
```

HLIM Hardware Limit Enable

Format	HLIM {*value*}
Group	Axis Limits
Units	See Below
Data Type	N/A
Default	0
Prompt Level	PROGx
See Also	HLBIT, HLDEC
Related Topics	N/A
Product Revision	Command and Firmware Release

This command enables or disables the inputs defined as hardware end-of travel limits. With limits disabled, motion is not restricted.

When you enable a specific limit (positive or negative) and the limit wiring for the enabled limit is a physical open circuit, motion is restricted—the defaults for the Limit Level Invert (bits 20 and 21, Parameters 4600-4615) bits are set.

NOTE: If limits are enabled and the motor/load encounters a hardware limit, motion stops at the rate set by the **HLDEC** command. To clear the switch, motion must occur in the opposite direction.

HLIM **Hardware Limits**

Value	Description
0	Disables positive limit and negative limit (default)
1	Enables positive limit and disables negative limit
2	Disables positive limit and enables negative limit
3	Enables positive limit and negative limit

Example

The following example enables both positive and negative hardware limits for axis X.

```
HLIM X3
```

HSINT High-Speed Interruptible Move

Format	HSINT *axis*(*mode*, *target*, *incmove*{, *window*{, *wstart*{, *abortbit*}}}) {CAP *capture_register*}
Group	Feedback Control
Units	Units based on PPU
Data Type	Target-FP32, Incmove-FP32
Default	N/A
Prompt Level	PROGx
See Also	IHPOS, INT, INTCAP, MSEEK, PPU
Related Topics	Axis Flags (0-7)(8-15),
Product Revision	Command and Firmware Release

This command initiates a high-speed interruptible move. The **HSINT** sequence consists of an incremental or absolute move with a capture window in the middle of it. Within the capture window, an internal **INTCAP** is initiated and monitored. If a capture occurs within the window, the current move is killed and a second move is started.

The mode parameter and hardware capture register are the same as those used in the **INTCAP** command. Refer to the **INTCAP** command for mode parameter and hardware capture register selection.

The components of the **HSINT** command are as follows:

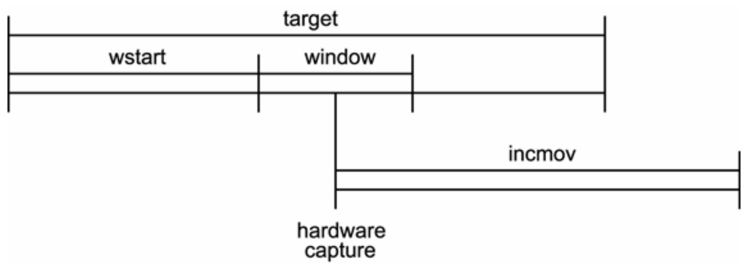
- *axis:* can be either incremental when using a trailing slash mark (/), or absolute. A master can only execute an **HSINT** command for a single axis at a time.
- *mode:* same as the mode in the **INTCAP** command and is used to start looking for a hardware capture in the capture window.
- *target:* starts the incremental or absolute move indicated by the *axis* argument. This move drives the rest of the **HSINT** sequence.
- *incmove:* added to the capture position if a hardware capture is detected within the capture window. In this case, the axis "**HSINT** Registered" flag is set, the move in progress is killed, and this new value is used as an absolute target position for the rest of the **HSINT** sequence.
- *window:* defines the width of the capture window. If this parameter is not present or is zero, the capture window is then defined as the area between the start of the window and the end of the move.
- *wstart:* defines the start of the capture window. If this parameter is not present, the window begins at the start of the move.
- *abortbit:* a flag that is monitored during the entire **HSINT** sequence. If the bit is seen, the current move is killed and the axis "HSINT Aborted" flag is set indicating a user abort condition.
- *capture_register:* same as the hardware capture register in the **INTCAP** command.

Program flow will continue to the next line/command after the "incmov" begins or after the end of the capture window has been passed. If, however, the “abortbit” is being monitored, program flow will continue only after the original move ends, the “incmov” ends, or the entire sequence is aborted. For an illustration of **HSINT**, see below.

In the Axis Flags, you can use the “HSINT Registered” and “HSINT Aborted” (bit index 10 and 11, respectively) flags to monitor the high speed move.

Operation Sequence

The following illustrates the **HSINT** operating sequence:



HSINT with Servo

The following examples assume ENC2 as position feedback on Axis0 (X) as follows:

```
ATTACH AXIS0 ENC2 DAC0 ADC0
```

The **HSINT** command starts an incremental **HSINT** sequence with a rising primary external capture input, a target move distance of 100000 units, a move after capture of 50000 units, a capture window with a width 20000 units starting 10000 units into the move, and monitoring input 9 for an external abort signal.

Example 1

```
100 HSINT X/(2,100000,50000,20000,10000,9)
```

Example 2 (version 1.18 00)

```
100 HSINT X/(2,100000,50000,20000,10000,9) CAP0
```

HSINT with Stepper (version 1.18 06)

Here is the procedure that needs to be followed for using **HSINT** command with stepper.

- Attach the axis with encoder feedback and stepper output.
- Set the secondary axis flag “ Encoder Bypass Servo Loop”.
- The stepper and encoder should have one to one ratio.

Example

```
ATTACH AXIS0 ENC0 STEPPER0 NONE
SET 2327 : REM Encoder Bypass Servo Loop
```

INT Response Period (version 1.18 04)

When the hardware capture occurs and the second move is started immediately, then there may be a small glitch in the motion since there is a finite time required to load the second incremental move. Master parameter “ INT Response Period” is added to avoid this scenario. This does not try to start the second move immediately. Rather the current move is extrapolated, while the second move is loaded into the buffer. Then after the INT Response Period, the moves are switched automatically and there is no glitch in motion. The draw back of this method is that it extrapolates the current move and will work well only when the master is at constant steady velocity when the capture occurs. If the contact velocity can't be guaranteed then this feature should not be used by setting the INT Response Period to –1. The default value for the INT Response period is 5 (the units are in servo period).

IDELAY Set Integral Time-Out Delay

Format	IDELAY {*axis* {*value*}} {*axis* {*value*}} ...
Group	Servo Control
Units	seconds
Data Type	FP32
Default	0
Prompt Level	PROGX
See Also	AXIS, IGAIN, ILIMIT
Related Topics	Axis Parameters (0-7) (8-15)
Product Revision	Command and Firmware Release

This command modifies the value used in the PID algorithm to control integral delay. The integral delay determines the amount of time, after a move ends, before integration begins. If the value is set to zero, integration is active all the time, even during moves.

Issuing an **IDELAY** command to an axis without an argument will display the current setting for that axis. The default gain is 0.0 for all axes.

Example

The following example sets the X axis integral time-out delay to 100 milliseconds:

```
IDELAY X0.1
```

IF/THEN Conditional Execution

Format	IF (*boolean*) THEN *command*
Group	Program Flow
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	BIT, IF/ELSE IF/ELSE/ENDIF
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

This command is used for conditional branching. If the Boolean expression is true, then the rest of the line is executed. Otherwise, the program drops down to the next line. The Boolean can either be an expression composed of several variables or a single bit reference.

Remarks

IF/THEN statements cannot be placed within an **IF/ENDIF** loop.

Examples

Example 1

The following example will set output 32 if input number 10 is active, and will clear 32 if bit 10 is off in a loop.

```
_LOOP1
IF (BIT 10) THEN SET 32
IF (NOT BIT 10) THEN CLR 32
GOTO LOOP1
```

Example 2

Inputs 3 through 6 are used to initiate moves for the X and Y Axes. A counter variable (P2) will increment each time a move is executed. The program will **END** when the number of moves exceeds a maximum value set in a variable (P3). Inputs 3 to 5 are normally-open switches, while input 6 is normally closed.

```
PROGRAM
P0=10   : REM  global variable used for move values
P2=0  : REM global variable used as counter
REM: for number of moves executed
P3=20   : REM global variable used for maximum number of moves
_procMainLoop
IF (BIT3) THEN P1=1 : GOSUB procXMove
IF (BIT4) THEN P1=-1:  GOSUB procXMove
IF (BIT5) THEN P1=1 : GOSUB procYMove
IF (NOT BIT6) THEN P1=-1 : GOSUB procYMove
IF (P2>P3) THEN END
GOTO procMainLoop

_ procXMove
MOV X/(P0*P1)
INH-516   : REM wait for move to complete
P2=P2+1
RETURN
```

```
_ procYMove
MOV Y/(P0*P1)   : REM incremental move for Y axis
INH-516   : REM wait for move to complete
P2=P2+1
RETURN
ENDP
```

IF/ELSE IF/ELSE/ENDIF Conditional Execution

Format	IF (*boolean*) ELSE IF (*boolean*) commands ELSE commands ENDIF
Group	Program Flow
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	BIT, DEFINE, DIM, IF/THEN
Related Topics	N/A
Product Revision	Command and Firmware Release

This command is used for conditional branching. If the boolean expression is true, then the rest of the line is executed. Otherwise, the program drops down to check the next boolean expression. The boolean can either be an expression composed of several variables or a single bit reference.. The Boolean can either be an expression composed of several variables or a single bit reference.

Example

The following example counts up if bit 32 is set, else if bit 33 is set it will count down. If neither bit is set, then it sets the counter to zero.

```
SYS
DIM PROG0(5000)
DIM DEF (10)
PROG0
#DEFINE COUNTER LV0
#DEFINE UPCOUNT BIT32
#DEFINE DOWN COUNT BIT33
PROGRAM
DIM LV2
_START
DWL 5
IF (UPCOUNT)
     COUNTER = COUNTER+1
     PRINT "Counting Up"
     PRINT "Counter = ", COUNTER
ELSE IF (DOWNCOUNT)
     COUNTER = COUNTER-1
     PRINT "Counting Down"
     PRINT "Counter = ", COUNTER
ELSE
     COUNTER = 0
     PRINT "Initializing"
     PRINT "Counter = ", COUNTER
ENDIF
GOTO start
ENDP
```

IGAIN Set Integral Gain

Format	IGAIN {*axis* {*value*}} {*axis* {*value*}} ...
Group	Servo Control
Units	volts / second / pulse
Data Type	FP32
Default	0
Prompt Level	PROGx
See Also	AXIS, DGAIN, FFACC, FFVEL, IDELAY, ILIMIT, PGAIN
Related Topics	Axis Parameters (0-7) (8-15)
Product Revision	Command and Firmware Release

This command modifies the value used in the PID algorithm to control integral gain. Issuing an **IGAIN** command to an axis without an argument will display the current setting for that axis. The default gain is 0.0 for all axes.

To reset the integral term component of the PID algorithm to zero:

- Turn **IGAIN** on by setting to a number other than zero.
- Set **ILIMIT** to zero.
- Set **IGAIN** to zero.

Example

The following example sets the X axis integral gain to 0.1 volts / second / pulse:

```
IGAIN X0.1
```

NOTE: If **ILIMIT** is zero than the integral will remain off, even if the **IGAIN** value is set to other than zero.

IHPOS Inhibit on Position

Format 1:	IHPOS + *parameter*(*setpoint*, *timeout*)
Format 2:	IHPOS - *parameter*(*setpoint*, *timeout*)
Group	Logic Function
Units	Units based on PPU
Data Type	FP32
Default	N/A
Prompt Level	PROGx
See Also	CLR, INH, SET, PPU
Related Topics	N/A
Product Revision	Command and Firmware Release

This command causes the program to inhibit (pause) further program execution until the specified parameter passes the given 'setpoint'. Although typically used to inhibit on axis position, this command can watch any system or user defined parameter.

The *timeout* argument sets a maximum time, in seconds, to wait for the condition to be met. If the condition is not met within this time limit, the program sets its timeout flag and continues normally. If the timeout is zero, there is no timeout checking done.

Issuing an **IHPOS** followed by a plus sign will inhibit until the parameter is greater than or equal to the setpoint. The minus sign will inhibit until the parameter is less than or equal to the setpoint. The plus sign is optional.

Example

The following example will inhibit until the position of ENC0 (P6144) is less than or equal to 10000 pulses, or 1.5 seconds have elapsed:

```
IHPOS -P6144(10000,1.5)
```

ILIMIT Set Integral Anti-Windup Limit

Format	ILIMIT {*axis* {*value*}} {*axis* {*value*}} ...
Group	Servo Control
Units	volts
Data Type	FP32
Default	0
Prompt Level	PROGx
See Also	AXIS, IDELAY, IGAIN
Related Topics	Axis Parameters (0-7) (8-15)
Product Revision	Command and Firmware Release

This command modifies the value used by the PID filter to limit the amount of integral term allowed to build up in the loop. Issuing an **ILIMIT** command to an axis without an argument will display the current setting for that axis. The default limit is 0.0 for all axes.

NOTE: The **ILIMIT** should be set to a value other than zero for the integrator to become operational.

Example

The following example sets the X axis integration limit to 0.5 volts:

```
ILIMIT X0.5
```

INH Inhibit on a Bit High or Low

Format 1:	INH {+} *index {(Timeout)}*
Format 2:	INH {-} *index {(Timeout)}*
Group	Logic Function
Units	N/A
Data Type	FP32 *(Timeout)*
Default	N/A
Prompt Level	PROGx
See Also	CLR, SET
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Causes the program to inhibit (pause) program execution until the specified bit is in the selected state, or until timeout occurs.

Remarks

The optional *timeout* argument sets a maximum time, in seconds, to wait for the condition to be met. If the condition is not met within this time limit, the program sets its timeout flag and continues normally. If the timeout is zero, there is no timeout checking done.

Argument

Format 1

INH {+} Selects the ON state of a bit.

Format 2

INH {-} Selects the OFF state of a bit.

Remarks

You can select the ON or OFF state of a bit as the trigger. To select the OFF state of a bit, you must include the minus sign in the command syntax.

You must put parentheses around variables used as arguments to the inhibit command; otherwise, a syntax error will result.

NOTE: Use caution when using inhibit statements in non-motion programs (PROG8 through 15 *). Inhibiting one non-motion program will inhibit all non-motion programs because they share a processor time slice.

* PROG1 through 15 for Aries Controller.

Aliases

Parentheses may or may not be required when using aliases with the inhibit command:

- Parentheses are not required when an alias is assigned to a constant or a Bit.
- When you assign an alias to a variable, you must put parentheses around the variable. It is useful to be descriptive when using aliases for bit checking as a reminder to use parentheses.

Examples

Example 1

The following program will inhibit until input 24 clears or evaluates as FALSE:

```
PROGRAM
PRINT "Waiting for input 24 to clear…"
INH -24
PRINT "Input 24 is clear."
ENDP
```

Example 2

The following program will inhibit until input 24 clears or evaluates as FALSE, or 1.5 seconds elapse:

```
PROGRAM
PRINT "Waiting for input 24 to clear…"
INH -24 (1.5)
PRINT "Input 24 is clear or timed out."
ENDP
```

Example 3

The following illustrates the use of parentheses when aliases are assigned to variables—as well as constants and Bits—and the usefulness of using descriptive names for variables:

```
#DEFINE BitOne BIT1
#DEFINE numberOne 1
#DEFINE varOne LV1

DIM LV10 : REM Dimension 10 long variables

REM Correct Usage
INH BitOne : REM Wait until bit 1 is active
INH numberOne : REM Wait until bit 1 is active
INH (varOne) : REM Wait until the bit referenced by LV1 is active

REM Incorrect Usage - generates "Syntax error"
INH varOne : REM Wait until the bit referenced by LV1 is active
```

Example 4

A common use of **INH** is to wait for motion to stop before executing the next command. Otherwise, the controller will continue to process commands while motion is in progress.

The program here does not use INH. In `GrabPart`, immediately following the start of `Z10` the gripper will close. The controller does not wait for the move to complete before `SET bOutGripper`. In `DropPart`, immediately following the start of `Z10` the one second dwell will begin. The dwell may complete prior to the move, then the gripper will open, dropping the part before the location is reached.

```
#DEFINE bOutGripper BIT32   : REM output 32 controls a gripper
PROGRAM
DRIVE ON X Y Z
DWL 0.2   : REM pause while drives energize
RES X Y Z   : REM set position counters to zero

_GrabPart
Z10   : REM move Z axis down to part location
SET bOutGripper   : REM  close gripper to grab a part
Z0

_MoveTo Drop
X10 Y10   : REM move X and Y to drop location

_DropPart
Z10
DWL 1   : REM pause 1 sec
CLR bOutGripper   : REM open gripper to release part
Z0   : REM move z back up

_ReturnToStart
X0 Y0   : REM return X and Y to start point
ENDP
```

Using `INH -516` (Master0 In Motion) will correct the sequence as follows:

```
#DEFINE bOutGripper BIT32   : REM output 32 controls a gripper
PROGRAM
DRIVE ON X Y Z
DWL 0.2   : REM pause while drives energize
RES X Y Z   : REM set position counters to zero

_GrabPart
Z10   : REM move Z axis down to part location
INH -516   : REM wait until any XYZ motion has completed before continuing
SET bOutGripper   : REM close gripper to grab a part
DWL 0.1   : REM pause 100msec to allow gripper to close
Z0

_MoveTo Drop
X10 Y10   : REM move X and Y to drop location

_DropPart
Z10
INH -516   : REM wait until any XYZ motion has completed before continuing
CLR bOutGripper   : REM open gripper to release part
DWL 0.1   : REM pause 100msec to allow gripper to open
Z0   : REM move z back up

_ReturnToStart
X0 Y0   : REM return X and Y to start point
ENDP
```

INPUT Receive Data from a Device

Format	INPUT {;} {#*device* ,} {"*prompt string*" *separator*} *parameterlist*
Group	Character I/O
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	CLOSE, OPEN, PRINT
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

This command receives data from a device and places the data into the designated parameters. If no device number is given, device zero is used. If the device is closed, or was never opened, the **INPUT** command will return an error.

When reached, the **INPUT** command pauses the program execution until input is received.

Remarks

The optional semicolon that follows the **INPUT** command controls the echo of characters as they are received. Characters are normally echoed. Placing a semicolon after the command will prevent the characters from being echoed.

If a "*prompt string*" is used, it will be printed out to the device before the parameters are read from the device. The separator after the prompt string can be either a comma (,) or a semicolon (;). If the separator is a semicolon, the final carriage return / linefeed output sequence will be suppressed. Otherwise, a carriage return/linefeed will be output after all of the data has been read from the device.

The "*parameterlist*" is a list of parameters separated with commas. When the data is read from the device, either a comma or a carriage return will cause the current field to be registered and the next field to begin. If the current field is the last parameter in the parameter list, the input command will end.

Characters less than CHR$(32) or greater than CHR$(126) will be ignored. In order to read these characters, the **INKEY$** function must be used.

Example 1

Opens communications to an external device with COM1 and serial connection. Controller sends strings to the device and waits for inputs back from the device.

```
PROGRAM
REM main program
DIM $V(1,80)
OPEN "COM1:9600,N,8,1" AS #1
PRINT #1,
PRINT #1, "Enter 'EXIT' to quit ..."
_LOOP1
```

```
INPUT #1, "Command?", $V0
$V0 = UCASE$($V0)
PRINT #1, "["; $V0; "]"
IF ($V0 = "EXIT") THEN GOTO LOOP2
GOTO LOOP1
_LOOP2
REM program shutdown
PRINT #1, "Program terminated"
CLOSE #1
ENDP
```

Example 2

This example simply prints messages to the ACR terminal emulator and waits for values back from the users. Execute this program in the terminal using the LRUN command to view the displayed messages.

```
PROGRAM
DIM $V(1,80)
PRINT,"Enter 'EXIT' to quit …"
_WaitForCommand
INPUT "Command?",$V0
$V0=UCASES$($V0)
PRINT,"[";$V0;"]"
IF ($V0="EXIT") THEN GOTO ShutDown
GOTO WaitForCommand
_ShutDown
REM program shutdown
PRINT, "Program terminated"
ENDP
```

INT Interruptible Move

Format	INT *sign index* {*axis*(*target*, *incmove*)} {*axis*(*target*, *incmove*)} ...		
Group	Interpolation		
Units	Units based on PPU		
Data Type	FP32		
Default	N/A		
Prompt Level	PROGx		
Report Back	None	**To view, type**	N/A
See Also	HSINT, MOV, PPU, SINE, TRJ		
Related Topics	Master Parameters (0-7) (8-15)		
Product Revision	Command and Firmware Release		

Description

Initiates an interruptible linear move..

Arguments

sign

Required. Indicates the state of the *index* argument being checked. Plus (+) is set. Minus (-) is clear.

index

Required. Assigns the bit to monitor.

axis

Optional. Assigns an axis.

target

Required. Position at the end of the move (in units).

incmove

Required. Incremental distance to move when the *sign* and *index* are true.

Remarks

The **INT** command monitors a specific bit while performing a move.

- While the *sign index* condition is false, the motion proceeds to the *target* position.
- If the *sign index* condition is true, the controller abandons the *target* move and initiates the *incmove* move.

NOTE: The *target* and *incmove* must move the same direction as currently traveling.

The coordinated motion profiler does not process any code statements below the **INT** statement until either the target position is achieved or the incremental move is initiated. As the current program is otherwise engaged with the **INT** statement, use another program to control the *sign index* condition—certain ACR series controllers use **INT** for homing routines.

You can also blend **INT** moves when **STP** is set to zero.

Incremental Moves

The incremental move must be long enough to allow for ramping down—as controlled by **VEL** and **STP** rates. Otherwise, the incremental move can stop without ramping down.

INT Response Period (version 1.18. 04)

The "INT Response Period" parameter improves the accuracy of the incremental move. In prior firmware versions, an unpredictable lag occurs between the time when the *sign index* condition becomes true and when the incremental move begins.

The "INT Response Period" parameter sets a response period (default is 5 servo period units). Before **INT** starts the *incmove*, the response period must finish. During that time the coordinated motion profiler loads the *incmove* and subtracts the distance traveled during the response period. The formula is as follows:

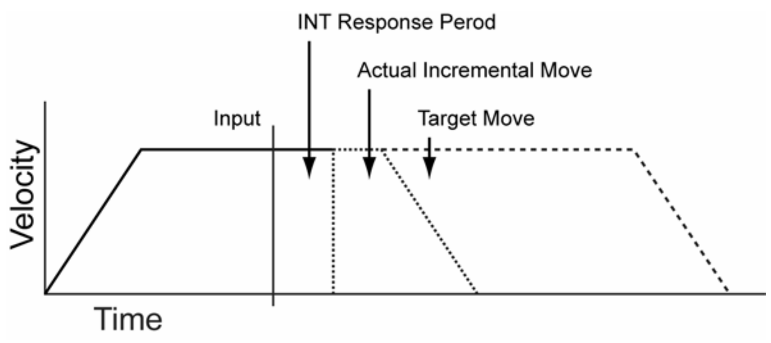
Actual incremental move= *incmove*-(v*INT response Period*servo update rate)

Where

v=current axis velocity

At the end of the response period, the incremental move begins. The extrapolated incremental move provides an accuracy previously not available.

NOTE: The function works only with a constant velocity. If you cannot guarantee a constant velocity, you can turn off the feature—set the "INT Response Period" to -1.



Example

The following example starts a move toward X100000. If the output bit 32 clears before the move completes, axis X moves and stops an incremental 2000 units from the trigger.

```
INT -32 X(100000,2000)
```

INTCAP Encoder Capture

Format	INTCAP {*axis mode* {*capture_register capture_parameter*}} {*axis mode* {*capture_register capture_parameter*}} ...
Group	Feedback Control
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	INTCAP OFF
Related Topics	Axis Flags (0-7) (8-15), Axis Parameters (0-7) (8-15)
Product Revision	Command and Firmware Release

Description

This command enables hardware position capture triggered from one of several different sources.

Each hardware platform has a specific number (from 1-18) of capture registers available. For each hardware capture register, there are several different capture sources. After the **INTCAP** command is issued, the next capture trigger (specified edge of capture source) causes the hardware to latch the encoder count of the position feedback encoder of the axis, stores the value in the desired capture register and sets an interrupt. Then an interrupt handler transfers the captured position from the capture register into the "hardware capture" parameter and sets the appropriate "capture complete" flag.

- Entering the command **INTCAP** without any arguments will list the currently armed capture registers.
- Multiple captures can be armed using a single command.
- The latency on the capture is less than 100 nanoseconds (1 microsecond delay for external input signals coming through the opto-isolators, 400 nanoseconds for the ACR9000.)
- No motion is commanded by, or generated as a result of, the **INTCAP** command; it is simply a mechanism to arm the capture to take place when a given source is triggered.
- The **INTCAP** command is also incorporated in multiple commands in the ACR controller firmware such as **HSINT** (High Speed Interruptible move), **MSEEK** (Marker Seek), and triggered gear and cam functions.

Arguments

Axis

Indicates which axis' encoder position will be captured. If the axis is using two encoders (dual loop feedback), the position encoder is used.

Mode

For each hardware capture register, there are several different capture sources: markers (Encoder Z Channels/reference marks) and external inputs. Both the rising and falling edges can be selected. See chart below for the Valid Sources for the specific hardware platform used. For example, Mode "0" will trigger a capture when the rising edge of the first marker (Encoder Z Channel or reference mark) is found.

Capture Register

Optional. Specifies which register is armed to hold the captured position.

- If the hardware capture register is not specified, the hardware capture register index is the same as the feedback encoder index of the axis used to enable the hardware capture. The capture complete flag and hardware capture parameter for the encoder capture is selected based on the axis used to enable the hardware capture.
- If the hardware capture register is specified then the capture parameter must also be specified. The capture complete flags and hardware capture parameters are still in pairs, but are not based on the axis used to enable the hardware capture.

Capture Parameter

Optional. Specifies which "hardware capture" parameter and "capture complete" flag is used to store the captured value after it is passed from the capture register.

Remarks

The following lists values for valid capture registers and modes by product name.

Valid Capture Registers

	9000 9030	**8020**	**1505**	**AR-xxPE**	**AR-xxCE**
CAP 0	ENC0-3, 8	ENC0-3, 8	ENC0-3	EPLDx	ENC0-1
CAP 1	ENC0-3, 8	ENC0-3, 8	ENC0-3	n/a	ENC0-1
CAP 2	ENC0-3, 8	ENC0-3, 8	ENC0-3	n/a	n/a
CAP 3	ENC0-3, 8	ENC0-3, 8	ENC0-3	n/a	n/a
CAP 4	ENC4-7, 9	ENC4-7, 9	n/a	n/a	n/a
CAP 5	ENC4-7, 9	ENC4-7, 9	n/a	n/a	n/a
CAP 6	ENC4-7, 9	ENC4-7, 9	n/a	n/a	n/a
CAP 7	ENC4-7, 9	ENC4-7, 9	n/a	n/a	n/a

Valid Modes

Mode	Source Description	9000 9030	8020	1505	AR-xxPE	AR-xxCE
0	Rising First Marker	Y	Y	Y	Y	Y
1	Rising Second Marker	Y	Y	Y	N	Y
2	Rising First External	Y	Y	Y	Y	Y
3	Rising Second External	Y	Y	Y	Y	Y
4	Rising First Marker	Y	Y	Y	Y	Y
5	Falling Second Marker	Y	Y	Y	N	Y
6	Falling First External	Y	Y	Y	Y	Y
7	Falling Second External	Y	Y	Y	Y	Y
8	Rising Third Marker	Y	Y	Y	N	N
9	Rising Fourth Marker	Y	Y	Y	N	N
10	Rising Third External	Y	Y	Y	Y	Y
11	Rising Fourth External	Y	Y	Y	N	Y
12	Falling Third Marker	Y	Y	Y	N	N
13	Falling Fourth Marker	Y	Y	Y	N	N
14	Falling Third External	Y	Y	Y	Y	Y
15	Falling Forth External	Y	Y	Y	N	Y
16	Rising Fifth Marker	Y	Y	N	N	N
17	Rising Sixth Marker	N	Y	N	N	N
18	Rising Fifth External	Y	Y	N	N	N
19	Rising Sixth External	Y	Y	N	N	N
20	Falling Fifth Marker	Y	Y	N	N	N
21	Falling Sixth Marker	N	Y	N	N	N
22	Falling Fifth External	Y	Y	N	N	N
23	Falling Sixth External	Y	Y	N	N	N
24	Rising Seventh Marker	N	Y	N	N	N
25	Rising Eighth Marker	N	Y	N	N	N
26	Rising Seventh External	Y	Y	N	N	N
27	Rising Eighth External	Y	Y	N	N	N
28	Falling Seventh Marker	N	Y	N	N	N
29	Falling Eighth Marker	N	Y	N	N	N
30	Falling Seventh External	Y	Y	N	N	N
31	Falling Eighth External	Y	Y	N	N	N
32	Rising Ninth Marker	N	Y	N	N	N
33	Rising Tenth Marker	N	Y	N	N	N
36	Falling Ninth Marker	N	Y	N	N	N
37	Falling Tenth Marker	N	Y	N	N	N

The following tables list capture sources by product name, where:

MRK	Encoder Marker (Z Channel)
INP	Input
HS INP	High Speed Input
EXP INP	Expansion Input

ACR9000 Capture Sources

Capture Register	Marker				
	First	Second	Third	Fourth	Fifth
CAP 0	MRK 0	MRK 1	MRK 2	MRK 3	MRK 8
CAP 1	MRK 1	MRK 0	MRK 3	MRK 2	MRK 8
CAP 2	MRK 2	MRK 3	MRK 0	MRK 1	MRK 8
CAP 3	MRK 3	MRK 2	MRK 1	MRK 0	MRK 8
CAP 4	MRK 4	MRK 5	MRK 6	MRK 7	MRK 9
CAP 5	MRK 5	MRK 4	MRK 7	MRK 6	MRK 9
CAP 6	MRK 6	MRK 7	MRK 4	MRK 5	MRK 9
CAP 7	MRK 7	MRK 6	MRK 5	MRK 4	MRK 9

Capture Register	External							
	First	Second	Third	Fourth	Fifth	Sixth	Seventh	Eighth
CAP 0	INP 24	INP 25	INP 26	INP 27	INP 28	INP 29	INP 30	INP 31
CAP 1	INP 25	INP 24	INP 27	INP 26	INP 29	INP 28	INP 31	INP 30
CAP 2	INP 26	INP 27	INP 24	INP 25	INP 30	INP 31	INP 28	INP 29
CAP 3	INP 27	INP 26	INP 25	INP 24	INP 31	INP 30	INP 29	INP 28
CAP 4	INP 72	INP 73	INP 74	INP 75	INP 76	INP 77	INP 78	INP 79
CAP 5	INP 73	INP 72	INP 75	INP 74	INP 77	INP 76	INP 79	INP 78
CAP 6	INP 74	INP 75	INP 72	INP 73	INP 78	INP 79	INP 76	INP 77
CAP 7	INP 75	INP 74	INP 73	INP 72	INP 79	INP 78	INP 77	INP 76

Aries Controller Capture Sources (ARxxCE)

Capture Register	Marker		External			
	First	Second	First	Second	Third	Fourth
CAP 0	MRK 0	n/a	HS INP 5	HS INP 6	HS INP 4	n/a
CAP 1	n/a	MRK 0	HS INP 6	HS INP 5	n/a	HS INP 4

Aries EPL Capture Sources (ARxxPE)

Capture Register	Marker	External		
	First	First	Second	Third
CAP 0	MRK 0	HS INP 5	HS INP 6	HS INP 4

ACR1505 Capture Sources

Capture Register	Marker				External			
	First	Second	Third	Fourth	First	Second	Third	Fourth
CAP 0	MRK 0	MRK 1	MRK 2	MRK 3	INP 0	INP 1	INP 2	INP 3
CAP 1	MRK 1	MRK 0	MRK 3	MRK 2	INP 1	INP 0	INP 3	INP 2
CAP 2	MRK 2	MRK 3	MRK 0	MRK 1	INP 2	INP 3	INP 0	INP 1
CAP 3	MRK 3	MRK 2	MRK 1	MRK 0	INP 3	INP 2	INP 1	INP 0

ACR8020 Capture Sources

Capture Register	Marker									
	First	Second	Third	Fourth	Fifth	Sixth	Seventh	Eighth	Ninth	Tenth
CAP 0	MRK 0	MRK 1	MRK 2	MRK 3	MRK 4	MRK 5	MRK 6	MRK 7	MRK 8	MRK 9
CAP 1	MRK 1	MRK 0	MRK 3	MRK 2	MRK 5	MRK 4	MRK 7	MRK 6	MRK 9	MRK 8
CAP 2	MRK 2	MRK 3	MRK 0	MRK 1	MRK 6	MRK 7	MRK 4	MRK 5	MRK 8	MRK 9
CAP 3	MRK 3	MRK 2	MRK 1	MRK 0	MRK 7	MRK 6	MRK 5	MRK 4	MRK 9	MRK 8
CAP 4	MRK 4	MRK 5	MRK 6	MRK 7	MRK 0	MRK 1	MRK 2	MRK 3	MRK 8	MRK 9
CAP 5	MRK 5	MRK 4	MRK 7	MRK 6	MRK 1	MRK 0	MRK 3	MRK 2	MRK 9	MRK 8
CAP 6	MRK 6	MRK 7	MRK 4	MRK 5	MRK 2	MRK 3	MRK 0	MRK 1	MRK 8	MRK 9
CAP 7	MRK 7	MRK 6	MRK 5	MRK 4	MRK 3	MRK 2	MRK 1	MRK 0	MRK 9	MRK 8

Capture Register	External							
	First	Second	Third	Fourth	Fifth	Sixth	Seventh	Eighth
CAP 0	INP 24	INP 25	INP 26	INP 27	INP 28	INP 29	INP 30	INP 31
CAP 1	INP 25	INP 24	INP 27	INP 26	INP 29	INP 28	INP 31	INP 30
CAP 2	INP 26	INP 27	INP 24	INP 25	INP 30	INP 31	INP 28	INP 29
CAP 3	INP 27	INP 26	INP 25	INP 24	INP 31	INP 30	INP 29	INP 28
CAP 4	INP 28	INP 29	INP 30	INP 31	INP 24	INP 25	INP 26	INP 27
CAP 5	INP 29	INP 28	INP 31	INP 30	INP 25	INP 24	INP 27	INP 26
CAP 6	INP 30	INP 31	INP 28	INP 29	INP 26	INP 27	INP 24	INP 25
CAP 7	INP 31	INP 30	INP 29	INP 28	INP 27	INP 26	INP 25	INP 24

ACR1200 Capture Sources

Capture Register	Marker				External			
	First	Second	Third	Fourth	First	Second	Third	Fourth
CAP 0	MRK 0	MRK 1	MRK 2	n/a	INP 12	INP 13	INP 14	n/a
CAP 1	MRK 1	MRK 0	n/a	MRK 2	INP 13	INP 12	n/a	INP 14
CAP 2	MRK 2	n/a	MRK 0	MRK 1	INP 14	n/a	INP 12	INP 13

EXPAXIS Module Capture Sources

Capture Register	Marker									
	First	Second	Third	Fourth	Fifth	Sixth	Seventh	Eighth	Ninth	Tenth
CAP 10	MRK 0	MRK 1	MRK 2	MRK 3	MRK 4	MRK 5	MRK 6	MRK 7	MRK 8	MRK 9
CAP 11	MRK 1	MRK 0	MRK 3	MRK 2	MRK 5	MRK 4	MRK 7	MRK 6	MRK 9	MRK 8
CAP 12	MRK 2	MRK 3	MRK 0	MRK 1	MRK 6	MRK 7	MRK 4	MRK 5	MRK 8	MRK 9
CAP 13	MRK 3	MRK 2	MRK 1	MRK 0	MRK 7	MRK 6	MRK 5	MRK 4	MRK 9	MRK 8
CAP 14	MRK 4	MRK 5	MRK 6	MRK 7	MRK 0	MRK 1	MRK 2	MRK 3	MRK 8	MRK 9
CAP 15	MRK 5	MRK 4	MRK 7	MRK 6	MRK 1	MRK 0	MRK 3	MRK 2	MRK 9	MRK 8
CAP 16	MRK 6	MRK 7	MRK 4	MRK 5	MRK 2	MRK 3	MRK 0	MRK 1	MRK 8	MRK 9
CAP 17	MRK 7	MRK 6	MRK 5	MRK 4	MRK 3	MRK 2	MRK 1	MRK 0	MRK 9	MRK 8

Capture Register	External							
	First	Second	Third	Fourth	Fifth	Sixth	Seventh	Eighth
CAP 10	EXP INP 0	EXP INP 1	EXP INP 2	EXP INP 3	EXP INP 4	EXP INP 5	EXP INP 6	EXP INP 7
CAP 11	EXP INP 1	EXP INP 0	EXP INP 3	EXP INP 2	EXP INP 5	EXP INP 4	EXP INP 7	EXP INP 6
CAP 12	EXP INP 2	EXP INP 3	EXP INP 0	EXP INP 1	EXP INP 6	EXP INP 7	EXP INP 4	EXP INP 5
CAP 13	EXP INP 3	EXP INP 2	EXP INP 1	EXP INP 0	EXP INP 7	EXP INP 6	EXP INP 5	EXP INP 4
CAP 14	EXP INP 4	EXP INP 5	EXP INP 6	EXP INP 7	EXP INP 0	EXP INP 1	EXP INP 2	EXP INP 3
CAP 15	EXP INP 5	EXP INP 4	EXP INP 7	EXP INP 6	EXP INP 1	EXP INP 0	EXP INP 3	EXP INP 2
CAP 16	EXP INP 6	EXP INP 7	EXP INP 4	EXP INP 5	EXP INP 2	EXP INP 3	EXP INP 0	EXP INP 1
CAP 17	EXP INP 7	EXP INP 6	EXP INP 5	EXP INP 4	EXP INP 3	EXP INP 2	EXP INP 1	EXP INP 0

ACR9000 Examples

In the following examples:

- AXIS0 "X" uses Encoder0 for position feedback
- AXIS1 "Y" uses Encoder1 for position feedback

```
PROGRAM
RES X Y : REM reset axis counter to zero
GOSUB Example1
GOSUB Example2
GOSUB Example3
GOSUB Example4
END

_Example1
INTCAP X0   : REM arms capture for X Axis, using Encoder0 reference marker
JOG FWD X : REM initiate jogging move on X axis
INH 777   : REM wait until  Axis0 flag “Capture Complete” is set
JOG OFF X : REM stop jogging
REM capture parameter was not specified in INTCAP command, defaults to
REM Axis0
PRINT P12292
RETURN

_Example2
REM use capture register 0, Mode2, first external input: ACR9000 Input 24
INTCAP X2   : REM arms capture for X Axis
JOG FWD X : REM initiate jogging move on X axis
INH 777   : REM wait until  Axis0 flag “Capture Complete” is set
JOG OFF X : REM stop jogging
```

```
REM capture parameter was not specified in INTCAP command, defaults to
REM Axis0
PRINT P12292
RETURN

_Example3
REM Y axis use capture register 1, Mode0, rising edge of Encoder1
REM reference marker
INTCAP Y0  : REM arms capture for Y Axis
JOG FWD Y : REM initiate jogging move on X axis
INH 809  : REM wait until  Axis1 flag "Capture Complete" is set
JOG OFF Y : REM stop jogging
REM capture parameter was not specified in INTCAP command, defaults to
REM Axis1
PRINT P12548
RETURN

_Example4
REM Multi Axis capture, both axes use falling edge of Encoder0 Z marker as
REM the source
INTCAP X4 Y5
REM arms capture0, mode4 falling first marker, capture1 falling second
REM marker
JOG FWD X Y : REM initiate jogging move on X & Y axis
INH 809 : INH 777  : REM wait until Axis1 flag "Capture Complete" is set
JOG OFF X Y : REM stop jogging
PRINT P12292
REM capture parameter was not specified in INTCAP command, defaults to
REM Axis0
PRINT P12548
RETURN
ENDP

PROG1
PROGRAM
AXIS0 RES : REM Resets Axis0
AXIS1 RES : REM Resets Axis1
GOSUB Example5
GOSUB Example6
GOSUB Example7
END

_Example5
AXIS0 INTCAP 10 CAP2 P12804
REM Mode10, CAP2: Rising 3rd External, CAP2 uses Input24
AXIS1 INTCAP 11 CAP3 P13060
REM Mode11, CAP3: Rising 4th External, CAP3 uses Input24
AXIS0 JOG FWD : AXIS1 JOG FWD
INH 841 : INH 873
PRINT "Axis0 Capture Position", P12804
PRINT "Axis1 Capture Position", P13060
RETURN

_Example6
AXIS0 INTCAP 10 CAP2 P12804
REM Mode10, CAP2: Rising 3rd External, CAP2 uses Input24
AXIS1 INTCAP 11 CAP3 P13060
REM Mode11, CAP3: Rising 4th External, CAP3 uses Input24
AXIS0 JOG FWD : AXIS1 JOG FWD
INH 841 : INH 873
PRINT "Axis0 Capture Position", P12804
PRINT "Axis1 Capture Position", P13060
RETURN

_Example7
AXIS0 INTCAP 18
REM Mode18, CAP0: Rising 5th External, CAP0 uses Input28
AXIS0 INTCAP 19 CAP1 P12548
REM Mode18, CAP1: Rising 5th External, CAP1 uses Input29
AXIS0 INTCAP 18 CAP2 P12548
REM Mode18, CAP2 : Rising 5th External, CAP2 uses Input30
```

```
AXIS0 JOG FWD
INH 777 : INH 809 : INH 841
PRINT "Axis0 Capture Positions", P12292, P12548, P12804
RETURN

ENDP
```

Notes

- ACR9000: When using the SSI (Synchronous Serial Interface) feedback devices, the SSI data is transmitted serially and cannot be immediately transferred to the firmware. The amount of time it takes to get an encoder position after a request from firmware is significant, and is a function of the clock speed at which the encoder transmits this data. It is not possible for the firmware to use the fresh data from the encoder, rather it will use the last received data from the encoder. The worst case latency could be up to one servo period. This is much better latency than trying to use the software capture. (See below for information on software capture.) However, this will not be as good as 100 nsec latency as on the quadrature incremental encoder.
- ACR8020 and ACR1505: If an Expansion I/O board is present, and the CONFIG IO and CONFIG XIO commands are used to redirect the I/O bits, the hardware capture external sources remain as the appropriate hardware input on the main boards. (For example, if an ACR8020/ACR1505 and Expansion I/O board are present, and the CONFIG IO and CONFIG XIO commands are used to redirect the I/O bits, the hardware capture external sources remain as hardware inputs 24 through 31 on the ACR8020 controller, and hardware inputs 0 through 3 on the ACR1505.)
- ACR8020, ACR9000, and ACR9030 only: If the feedback encoder is Encoder 8 or 9, the hardware capture register must be specified, and that encoder must be attached as the position feedback for an axis. See ATTACH AXIS.
- ACR8020 EXPAXIS only: If the feedback encoder is Encoder 18 or 19, the hardware capture register must be specified.

Software Capture

It is possible to initiate an encoder capture sequence through software. This is done by issuing the **SET113** command. This triggers the software capture at the next period. It captures all encoder position object parameters and transfers them into the Axis Software Capture Parameter by encoder number—this cannot be modified. For example, the Encoder0 parameter (P6144) is placed in Software Capture parameter P12293, and so on. On completion of the captures, Bit113 is automatically cleared.

Example

```
Set113
INH-113
PRINT "Encoder positions captured"
```

Using INTCAP Functions with Stepper Output

When using an ACR controller card with stepper output, using a command that incorporates the **INTCAP** command as part of its function requires some program modification due to the operation of that command.

Problem When Using Stepper Output

The **INTCAP** command uses the position feedback encoder of the **ATTACH AXIS** command to determine what encoder object is used. If the **ATTACH AXIS** command does not have an encoder input stated as the feedback object, the **INTCAP** command will return an "Encoder not attached in line xx" error message.

When the ACR controller card is used with stepper output, the **ATTACH AXIS** command sets the Stepper Count object as both the position feedback and output object.

Example

```
CONFIG ENC2 STEPPER2 NONE NONE
ATTACH AXIS0 STEPPER0 STEPPER0
ATTACH AXIS1 STEPPER1 STEPPER1
ATTACH MASTER0
ATTACH SLAVE0 AXIS0 "X"
ATTACH SLAVE1 AXIS1 "Y"
PPU X1 Y1
MULT X4 Y4
MSEEK X(10000,0)
PRINT "Marker Found"
END
```

When this program is LRUN, the following error will occur:

```
P01>LRUN
Encoder not attached in line 10
P01>
```

Fixing the Problem

To prevent this error, the **ATTACH AXIS** command must list an encoder as the position feedback object to allow the **INTCAP** command to function. This will then have the axis operate closed-loop using the position encoder as feedback. Normally, stepper motors are not used closed loop. To correct this problem, there are also two secondary axis control flags:

- Stepper Feedback—this flag (Bit Index 23 of Secondary Axis Flags) allows the axis to continue as an open loop stepper axis but allows the **INTCAP** to capture the value of an encoder attached to a stepper axis. This is commonly used when an encoder is used with the MSEEK command to establish an accurate "home" position.
- Do Not Use the Capture Register—this flag (Bit Index 22 of Secondary Axis Flags) is set in addition to the previous flag if there is no encoder attached to the axis. When this flag is set, after the **INTCAP** interrupt occurs the capture register is not used. The current position register value is used as it normally would when using an open loop stepper axis. This is commonly used when the **INTCAP** command is used to determine an accurate "registration" point when used with the **HSINT** command.

Example

Change CONFIG command and set Stepper Feedback secondary axis control flag for axes to be used. Subsequently, the program will operate successfully.

```
CONFIG ENC2 STEPPER2 NONE NONE
ATTACH AXIS0 ENC0 STEPPER0
ATTACH AXIS1 ENC1 STEPPER1
SET2327 : REM Set Stepper Feedback for Axis 0
SET2359 : REM Set Stepper Feedback for Axis 1
ATTACH MASTER0
ATTACH SLAVE0 AXIS0 “X”
ATTACH SLAVE1 AXIS1 “Y”
PPU X1 Y1
MULT X4 Y4
MSEEK X(10000,0)
INH-516
IF (Bit777) THEN PRINT “Marker Found”
END
```

INTCAP OFF INTCAP is Turned Off

Format	INTCAP OFF {*axis*}
Group	Feedback Control
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	AXIS, INTCAP
Related Topics	N/A
Product Revision	Command and Firmware Release

If the Intcap mode is enable and the user wants to turn it off before the trigger happens then this command can be used to turn off the Intcap.

Example

```
INTCAP X OFF
```

INVK Inverse Kinematics

Format	INVK {PROG *number*}
Group	Interpolation
Units	N/A
Data Type	N/A
Default	-1
Prompt Level	PROGx
See Also	INVK OFF, INVK ON
Related Topics	Axis Parameters (0-7) (8-15)
Product Revision	Command and Firmware Release

This command defines which program contains the inverse kinematics algorithm, and is used in conjunction with other commands.

The following is a list of valid **INVK** command combinations:

INVK ON: Enable inverse kinematics.

INVK OFF: Disable inverse kinematics.

You create the inverse kinematics algorithms to suite your machine's specific configuration. The algorithms can consist of equations, logical expressions, or other commands and construction implemented through the AcroBASIC Language.

Example

The following example results in a circle instead of a straight line. Program seven contains the algorithms to transform data, and program zero defines the motion profile. Notice that program zero does the following: defines the motion profile; declares which program contains the inverse kinematics algorithms; and enables the algorithm.

```
PROG7
PROGRAM
P12361= sin(P12360) : REM Describe transformation in PROG7
P12617= cos(P12360) : REM Describe transformation in PROG7
ENDP

PROG0
ATTACH MASTER0
ATTACH SLAVE0 AXIS0 "X"
ATTACH SLAVE1 AXIS1 "Y"
PPU X 2000 Y 2000 : REM Scale commands to engineering units
ACC 100 DEC 100 STP 0 VEL 0

INVK PROG7 : REM Tell MASTER0 where the transformation are
INVK ON : REM Turn on the Kinematics

PROGRAM0_start
X / 0.2 : REM Incremental move in Cartesian space
GOTO start
```

Downloading this motion program would result in a circle instead of a straight line because of the transformation described in program 7 (PROG7).

INVK OFF Inverse Kinematics

Format	INVK ON
Group	Interpolation
Units	N/A
Data Type	N/A
Default	-1
Prompt Level	PROGx
See Also	INVK, INVK ON
Related Topics	N/A
Product Revision	Command and Firmware Release

This command turns off the inverse kinematics.

Example

The following example disables inverse kinematics:

```
INVK OFF
```

INVK ON Inverse Kinematics

Format	INVK ON
Group	Interpolation
Units	N/A
Data Type	N/A
Default	-1
Prompt Level	PROGx
See Also	INVK, INVK OFF
Related Topics	N/A
Product Revision	Command and Firmware Release

This command turns on the inverse kinematics.

Example

The following example enables inverse kinematics:

```
INVK ON
```

IPB Set In-Position Band

Format	IPB {*axis* {*value*}} {*axis* {(*value1*, *value2*)}} ...
Group	Axis Limits
Units	Units based on PPU
Data Type	FP32
Default	0,0
Prompt Level	PROGx
See Also	AXIS, EXC, PPU
Related Topics	Axis Flags (0-7)(8-15), Axis Parameters (0-7) (8-15), Master Flags (0-7) (8-15)
Product Revision	Command and Firmware Release

Description

Sets the following-error limits monitored by the "not in-position" flags.

Remarks

When the following error of a given axis is outside of its in-position band, the appropriate flag is set. Otherwise, the flag is cleared. For masters, the flag is set if any of its slaves are outside of their in-position bands.

Issuing the **IPB** command to an axis without an argument displays the current positive and negative limits for that axis. Issuing the command with a single argument sets the positive limit to "value" and the negative limit to minus "value". Issuing the command with two arguments sets the positive limit to "value1" and the negative limit to "value2". The default for both is 0.0 for all axes.

Example 1

The following example sets an in-position band of ±0.5 units for X, Y and Z axes.

```
IPB X0.5 Y0.5 Z0.5
```

Example 2

The following example sets an in-position band of +3 and –1 units for the X axis.

```
IPB X(3,-1)
```

IP IP Address

Format	IP *command* {*"address_in_doted_notation"*}
Group	Operating System
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS
See Also	IP MASK
Related Topics	N/A
Product Revision	Command and Firmware Release

This command sets the IP address of a controller with the Ethernet option. You can also use the **IP** command to view the IP address and subnet mask assigned to a controller.

The following is a list of valid IP address command combinations.

IP MASK: Defines the subnet mask.

IP GATE: Defines the subnet gateway.

To change the IP address of a controller using the IP command, you must enclose the new IP address in quotes—for example **IP** *"xxx.xxx.xxx.xxx"*. To save the change to flash memory, issue an **ESAVE** command. You must reboot the controller for the change to take affect.

NOTE: The IP address is stored in the flash memory and is automatically restored at power up. Do not set the IP address or subnet mask to values that are incompatible with your network; you will loose communication with the controller.

Example 1

The following example reports the IP address and subnet mask for the controller

```
SYS>ip
IP      "172.25.8.22"
IP MASK "255.255.0.0"
```

Example 2

The following example sets a new IP address for the controller.

```
SYS>ip
IP      "172.25.8.01"
IP MASK "255.255.0.0"
```

IP MASK IP Mask

Format	IP MASK {*"address_in_dotted_notation"*}
Group	Operating System
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS
See Also	IP, IP GATE
Related Topics	N/A
Product Revision	Command and Firmware Release

This command sets the IP Mask of a controller with the Ethernet option.

NOTE: The IP mask is stored in the flash memory and is automatically restored at power up. Do not set the IP address or subnet mask to values that are incompatible with your network; you will lose communication with the controller.

To change the IP Mask of a controller using the **IP MASK** command, you must enclose the new IP mask in quotes—for example **IP MASK** *"xxx.xxx.xxx.xxx"*. To save the change to flash memory, issue an **ESAVE** command. You must reboot the controller for the change to take affect.

Example

The following example sets the subnet mask for the controller.

```
IP MASK "255.255.255.0"
```

IP GATE IP Gate

Format	IP GATE {*"address_in_dotted_notation"*}
Group	Operating System
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS
See Also	IP, IP MASK
Related Topics	N/A
Product Revision	Command and Firmware Release

This command sets the IP Gateway address for a controller with the Ethernet option. The gateway IP is the IP address on a local network that allows the controller to send packets beyond its own subnet.

NOTE: The IP gateway address is stored in flash memory and is automatically restored at power up.

To change the IP Gateway address for a controller using the **IP GATE** command, you must enclose the address in quotes—for example **IP GATE** *"xxx.xxx.xxx.xxx"*. To save the change to flash memory, issue an **ESAVE** command. You must reboot the controller for the change to take affect.

Example

The following example sets the gateway address for the controller.

```
SYS>ip gate "172.26.135.254"
```

ITB Set In-Torque Band

Format	ITB {*axis* {*value*}} {*axis* {(*value1*, *value2*)}} ...
Group	Axis Limits
Units	volts
Data Type	FP32
Default	0,0
Prompt Level	PROGx
See Also	AXIS, TLM
Related Topics	Axis Flags (0-7)(8-15), Master Flags (0-7) (8-15)
Product Revision	Command and Firmware Release

This command sets the voltage limits monitored by the "not in-torque band" flags. When the output voltage of a given axis is outside of its in-torque band, the appropriate flag is set. Otherwise, the flag is cleared. For masters, the flag is set if any of its slaves are outside of their in-torque bands.

The **ITB** only defines flag monitoring boundaries, it does not affect the analog output in any way. See the TLM command for information on physical output clipping.

Issuing the **ITB** command to an axis without an argument displays the current positive and negative limits for that axis. Issuing the command with a single argument sets the positive limit to "value" and the negative limit to minus "value". Issuing the command with two arguments sets the positive limit to "value1" and the negative limit to "value2". The default for both is 0.0 for all axes.

Example 1

The following example sets the torque band to 0.3 volts for X and Y.

```
ITB X0.3 Y0.3
```

Example 2

The following example sets the torque band to +3 and –1 volts for the X axis.

```
ITB X(3,-1)
```

IVEL Set Initial Velocity

Format	IVEL {*rate*}
Group	Velocity Profile
Units	units/second based on PPU
Data Type	FP32
Default	0
Prompt Level	PROGx
See Also	ACC, DEC,FVEL, MASTER, PPU, STP, VEL
Related Topics	N/A
Product Revision	Command and Firmware Release

This command sets the initial velocity value for a master move profile. If this value is zero (default) it is ignored. Otherwise, the move will start at this velocity regardless of the current **ACC** and **DEC** settings.

Issuing an **IVEL** command without an argument will display the current setting. The default initial velocity is zero. An error will be returned if no master is attached.

Example

The following example sets the initial velocity to 1000 units/second:

```
IVEL 1000
```

JLM Set Jog Limits

Format	JLM {*axis* {limit}} {*axis* {(*plus*, *minus*)}} ...
Group	Axis Limits
Units	Units
Data Type	FP32
Default	0,0
Prompt Level	PROGx
See Also	ALM, AXIS, BLM
Related Topics	Axis Parameters (0-7) (8-15)
Product Revision	Command and Firmware Release

This command sets the jog limits for an axis. The jog limits are only checked when the "jog limit check" bit is set and the **JOG FWD** or **JOG REV** commands are in operation. The **JOG ABS** and **JOG INC** commands ignore jog limits even if the "jog limit check" bit is set. Jog limits only place limits on jog offset calculations. The primary and secondary setpoint are not part of the jog limits.

When the "jog limit check" bit is set, the **JOG FWD** command will jog to the positive jog limit and stop. If the current jog offset is greater than the positive jog limit, the **JOG FWD** command will do nothing. Likewise, the **JOG REV** command will jog to the negative jog limit and if the offset is less than the negative jog limit, **JOG REV** will do nothing.

Issuing the **JLM** command to an axis without an argument displays the current positive and negative limits for that axis. Issuing the command with a single argument sets the positive limit to "value" and the negative limit to minus "value". Issuing the command with two arguments sets the positive limit to "value1" and the negative limit to "value2". The default for both is 0.0 for all axes.

Example

The following example sets the X axis jog limits to +3.5 and -1.0 units:

```
JLM X(3.5,-1.0)
```

JOG Single Axis Velocity Profile

Format	JOG *command* {*axis* {*data*}} {*axis* {*data*}} ...
Group	Setpoint Control
Units	DATA= UNITS BASED ON PPU
Data Type	DATA= FP32
Default	N/A
Prompt Level	PROGx
See Also	AXIS, BKL, BSC, CAM, GEAR, HDW, JLM, PPU
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Initializes and controls single axis velocity profiling (jogging).

Arguments

Jogging, used along with a second command, sets up an individual velocity profile for an axis based on the current jog parameters. This profile ramps to a given velocity, generating a jog offset.

The jog offset is used during the summation of the primary setpoint to generate motion, or to modify Coordinated Moves, Cam, or Gear profiles.

As in most commands, if replacing a constant with a variable or parametric equation, use parentheses to "contain" the variable/equation.

The following is a list of valid jog command combinations:

JOG ABS: Jog to absolute jog offset position.

JOG ACC: Set jog acceleration.

JOG DEC: Set jog deceleration.

JOG FWD: Jog axis forward.

JOG HOME: Set jog home.

JOG HOMVF: Set jog home final velocity.

JOG INC: Jog to incremental jog offset position.

JOG JRK: Set jog jerk.

JOG OFF: Stop jogging axis.

JOG REN: Transfer current position (coordinated moves offset) into jog offset.

JOG RES: Move jog offset into current position (coordinated moves offset).

JOG REV: Jog axis backward.

JOG SRC: Set external time base.

JOG VEL: Set jog target velocity.

Example 1

The following sets motion parameters, starts motion, waits for input 1 to go low and then stops motion.

```
JOG ACC X100 Y100
JOG DEC X100 Y100
JOG VEL X10 Y20
JOG FWD X Y : REM START MOTION
INH-1 : REM WAIT FOR INPUT ONE TO GO LOW
JOG OFF X Y : REM STOP MOTION
```

Example 2

The following jogs AXIS 2 to an absolute jog position of 2.5 units.

Instead of using axis names, such as X or Y, the following uses the AXIS command. Note that the **AXIS** command precedes the **JOG ABS** command.

```
AXIS2 JOG ABS 2.5
```

NOTE: The Primary Setpoint is the summation of the Coordinated Moves, Cam, Gear, and Jog offsets. The Secondary Setpoint is the summation of the Primary Setpoint and the Ballscrew and Backlash offsets. The Secondary Setpoint is the value that is used by the servo loop.

JOG ABS Jog to Absolute Position

Format	JOG ABS {*axis* {*target*}} {*axis* {*target*}} ...
Group	Setpoint Control
Units	Units based on PPU
Data Type	Target= FP32
Default	N/A
Prompt Level	PROGx
See Also	AXIS, BKL, BSC, CAM, GEAR, HDW, JLM, PPU
Related Topics	Axis Flags (0-7) (8-15)
Product Revision	Command and Firmware Release

This command will use the current jog settings to jog an axis to an absolute jog offset as indicated by the ***target*** argument. This motion is independent from the attached master and can run on top of the current motion profile.

The **JOG REN** command may be used before **JOG ABS** to transfer the current position into the jog offset. The **JOG RES** command may be used after **JOG ABS** to transfer the jog offset back into the current position.

Example

The following example jogs the X and Y jog offsets to (1.25, 2.50) units:

```
JOG ABS X1.25 Y2.50
```

JOG ACC Set Jog Acceleration

Format	JOG ACC {*axis* {accel}} {*axis* {accel}} ...
Group	Setpoint Control
Units	units/second2 based on PPU
Data Type	FP32
Default	0
Prompt Level	PROGx
See Also	AXIS, BKL, BSC, CAM, GEAR, HDW, JLM, PPU
Related Topics	Axis Parameters (0-7) (8-15)
Product Revision	Command and Firmware Release

This command sets the programmed jog acceleration for an axis. The jog acceleration is the ramp used when the current jog velocity is lower than the programmed value.

Issuing a **JOG ACC** command to an axis without an argument will display the current setting for that axis. The default jog acceleration is 0.0 for all axes.

Example

The following example sets the X axis jog acceleration to 20000 units/second^2:

```
JOG ACC X20000
```

JOG DEC Set Jog Deceleration

Format	JOG VEL {*axis* {*decel*}} {*axis* {*decel*}} ...
Group	Setpoint Control
Units	units/second2 based on PPU
Data Type	FP32
Default	0
Prompt Level	PROGx
See Also	AXIS, BKL, BSC, CAM, GEAR, HDW, JLM, PPU
Related Topics	Axis Parameters (0-7) (8-15)
Product Revision	Command and Firmware Release

This command sets the programmed jog deceleration for an axis. The jog deceleration is the ramp used when the current jog velocity is higher than the programmed value. It is also used when the **JOG OFF** command is issued.

Issuing a **JOG DEC** command to an axis without an argument will display the current setting for that axis. The default jog deceleration is 0.0 for all axes.

Example

The following example sets the X axis jog deceleration to 20000 units/second^2:

```
JOG DEC X20000
```

JOG FWD Jog Axis Forward

Format	JOG FWD {*axis*} {*axis*} ...
Group	Setpoint Control
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	AXIS, BKL, BSC, CAM, GEAR, HDW, JLM, JOG
Related Topics	Axis Flags (0-7) (8-15)
Product Revision	Command and Firmware Release

This command initiates a ramp to the velocity programmed with the set by the **JOG VEL** command.

This command works identically to the "jog forward" bit.

Example

The following example starts the X and Y axis jogging in the positive direction:

```
JOG FWD X Y
```

JOG HOME Go Home

Format	JOG HOME {*axis* {*direction*}} {*axis* {*direction*}} ...
Group	Setpoint Control
Units	1= positive direction, -1= negative direction
Data Type	N/A
Default	
Prompt Level	PROGx
See Also	JOG HOMVF
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

This command instructs the controller to search for the home position in the direction and on the axes specified by the command. If an end-of-travel limit is activated while searching for the home limit, the controller reverses direction and searches for home in the opposite direction. However, if the motor/load encounters a second end-of-travel limit, after the change of direction, the homing operation is aborted.

Remarks

The status of the homing operation is stored in bit indexes 6 and 7 of Parameters 4600-4615. With a successful homing operation, the controller issues a RES to set the axis position registers to zero.

Flag Parameter		4600	4601	4602	4603	4604	4605	4606	4607
		AXIS Number							
Status Flags	Bit Index	0	1	2	3	4	5	6	7
Found Home	6	16134	16166	16198	16230	16262	16294	16326	16358
Failed to find Home	7	16135	16167	16199	16231	16263	16295	16327	16359

Flag Parameter		4608	4609	4610	4611	4612	4613	4614	4615
		AXIS Number							
Status Flags	Bit Index	8	9	10	11	12	13	14	15
Found Home	6	16390	16422	16454	16486	16518	16550	16582	16614
Failed to Find Home	7	16391	16423	16455	16487	16519	16551	16583	16615

After receiving the JOG HOME command, the controller will start the move and continue processing subsequent commands. The Found Home and Failed To Find Home flags can be used by a program to wait until the home process is completed before issuing other move commands.

Examples

Example 1

The following example sets the homing directions as positive for axis X, and negative for axis Y.

```
JOG HOME X1 Y-1
```

Alternatively, the generic axis designations can be used:

```
AXIS0 JOG HOME 1
AXIS1 JOG HOME -1
```

Example 2

```
HLIM X3    : REM enable limits
JOG ACC X250    : REM set jog accel for homing
JOG DEC X250    : REM set jog decel for homing
JOG VEL X50    : REM set jog velocity for homing
JOG HOMVF X25    : REM set jog final velocity for homing
SET 16152    : REM Backup to edge is enabled
CLR 16153    : REM Backup to positive edge
CLR 16154    : REM set Final approach direction is positive
JOG HOME X1    : REM start homing in positive direction
REM Home Successful: BIT16134
REM Home Failed: BIT16135
WHILE (NOT BIT 16134)
IF (BIT16135)
        PRINT "Homing failed"
        END    : REM end the program
        END IF
WEND
REM axis is now at absolute zero
JOG ABS X100   : REM jog move to 100 unit position
```

Example 3

The motor is connected to a 5 millimeter per revolution ballscrew. The Axis is first positioned to the home sensor using the **JOG HOME** command. Then the **MSEEK** command is used to find the encoder Z mark more accurate positioning. The **MSEEK** commands include an incremental move value during which the controller is looking for the Z channel. The incremental move is set to 6 mm to ensure that the motor travels at least one full revolution.

```
JOG ACC X250    : REM set jog accel for homing
JOG DEC X250    : REM set jog decel for homing
JOG VEL X50    : REM set jog velocity for homing
JOG HOMVF X5    : REM set jog final velocity for homing
SET 16152    : REM Backup to edge is enabled
CLR 16153    : REM Backup to positive edge
CLR 16154    : REM set Final approach direction is positive
JOG HOME X1
REM Home Successful: BIT16134, Home Failed: BIT16135
WHILE (NOT BIT 16134)
        IF (BIT16135)
                PRINT "Homing failed"
                END    : REM end the program
                END IF
        WEND
ACC 500 DEC 500 STP 500 VEL 50
MSEEK X(6,0)
```

Example 4

A linear motor stage includes end of travel limits but does not have a home sensor. The encoder Z mark will be used for homing. The **JOG HOME** command will allow the motor to traverse the entire length of travel between the limits as the controller looks for the Z mark.

```
INTCAP X0    : REM arm the capture register
    REM Capture Mode=0: Rising First Marker, Z Mark for ENC0
    REM Hardware Capture Parameter : P12292
    REM Capture Complete Flag : BIT777
RES X    : REM Reset all position counters to zero
JOG ACC X1000
JOG DEC X1000
JOG VEL X100
JOG HOME X1    : REM Use jog home function to bounce off both limits
WHILE (NOT BIT 777 AND NOT BIT 16135)
```

```
    REM Wait until both limits are hit or marker is found
    REM BIT 16135: Failed to find Home
WEND
IF (BIT 16135) THEN GOTO procHomeFailed
JOG OFF X   : REM marker found, stop jogging
INH -792   : REM wait until jogging is stopped
JOG ABS X(P12292/P12375)
REM absolute jog move back to captured position
INH -792   : REM wait until jogging is stopped
RES X
RETURN

_procHomeFailed
REM try again or end program if home failed
END
```

JOG HOMVF Home Final Velocity

Format	JOG HOMVF {*axis* {*units*}} {*axis* {*units*}} …
Group	Setpoint Control
Units	Units
Data Type	FP32
Default	0
Prompt Level	PROGx
See Also	JOG HOME
Related Topics	N/A
Product Revision	Command and Firmware Release

This command specifies the velocity to use when the homing operation makes the final approach. To use this command you must set the Home Backup Enable bit (bit 24, Parameters 4600-4615).

The **JOG HOMVF** can be saved using the **ESAVE** command.

Example

The following example sets the homing velocity as 0.2 for axis X, and 0.3 for axis Y.

```
JOG HOMVF X0.2 Y0.3
```

JOG INC Jog an Incremental Distance

Format	JOG INC {*axis offset*} {*axis offset*} ...
Group	Setpoint Control
Units	Units based on PPU
Data Type	Offset= FP32
Default	N/A
Prompt Level	PROGx
See Also	AXIS, BKL, BSC, CAM, GEAR, HDW, JLM, PPU
Related Topics	N/A
Product Revision	Command and Firmware Release

This command will use the current jog settings to jog an axis an incremental distance from the current jog offset as indicated by the *offset* argument. This motion is independent from the attached master and can run on top of the current motion profile.

The **JOG REN** command may be used before **JOG INC** to transfer the current position into the jog offset. The **JOG RES** command may be used after **JOG INC** to transfer the jog offset back into the current position.

Examples

The following subsequent examples assume the following

```
ATTACH MASTER0
ATTACH SLAVE0 AXIS0 "X"
ATTACH SLAVE1 AXIS1 "Y"
#DEFINE pi P0
pi = 3.141592
DIM LV10
DIM DV10
```

Example 1

The following code snippets are given at the program prompt or in program where the axis is attached:

```
JOG INC X1
JOG INC X-1
```

Example 2

The following code snippets are given at any prompt or program:

```
AXIS0 JOG INC 1
AXIS0 JOG INC -1
```

Example 3

The following code snippets demonstrate the use of variables:

```
JOG INC X(LV2)
JOG INC AXIS0 (P0**2)
AXIS0 JOG INC (DV3)
P0 = 1 : REM P0 is variable for axis number
P1 = -1 : REM P1 is variable for incremental move distance
JOG INC AXIS(P0) (P1) : REM jog axis1 incrementally 10 units negative
JOG INC AXIS(P0-1) (P1) : REM jog axis0 incrementally 10 units negative
JOG INC AXIS(1-1) (-10) : REM jog axis0 incrementally 10 units negative
```

Example 4

The following code snippets demonstrate the use of aliases:

```
JOG INC AXIS0 (pi)
```

Example 5

The following code snippets demonstrates jogging multiple axes:

```
JOG INC X-1 Y-2
JOG INC AXIS0 (-1) AXIS1 (-2)
AXIS0 JOG INC -1
AXIS1 JOG INC -2
```

JOG JRK Set Jog Jerk (S-curve)

Format	JOG JRK {*axis* {*jerk*}} {*axis* {*jerk*}} ...
Group	Setpoint Control
Units	units/second3 based on PPU
Data Type	FP32
Default	0
Prompt Level	PROGx
See Also	AXIS, BKL, BSC, CAM, GEAR, HDW, JLM, PPU
Related Topics	Axis Parameters (0-7) (8-15)
Product Revision	Command and Firmware Release

This command controls the slope of the acceleration versus time profile. If jerk is zero, the acceleration profile is rectangular (linear acceleration). Otherwise, the acceleration profile is trapezoidal, clipped on top or bottom by the current **JOG ACC** and **JOG DEC** settings.

Issuing a **JOG JRK** command to an axis without an argument will display the current setting for that axis. The default jog jerk is 0.0 for all axes.

Example

The following example sets the X axis jog deceleration to 80000 units/second^3:

```
JOG JRK X 80000
```

JOG OFF Stop Jogging Axis

Format	JOG OFF {*axis*} {*axis*} …
Group	Setpoint Control
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	AXIS, BKL, BSC, CAM, GEAR, HDW, JLM, JOG
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

This command initiates a ramp down to zero speed.

Examples

Example 1

The following example stops jogging of the X, Y and Z axes:

```
JOG OFF X Y Z
```

Example 2

```
PROG0
PROGRAM
IF (BIT0)
        SET 128
        P2=35
        RUN PROG1
        ENDIF
IF (BIT1)
        SET 129
        P2=40
        RUN PROG1
        ENDIF
ENDP

PROG1
PROGRAM
AXIS0 JOG ACC 500
AXIS0 JOG VEL(P2)
AXIS0 JOG DEC 500
IF (BIT128) THEN GOTO procSpinner1
IF (BIT129) THEN GOTO procSpinner2
_procSpinner1
AXIS0 JOG FWD   : REM start jogging forward for X Axis
DWL 5   : REM wait 5 seconds before stopping
AXIS0 JOG OFF   : REM stop jogging
END

_procSpinner2
AXIS0 JOG FWD   : REM start jogging forward for X Axis
REM BIT 794  Primary Axis Flag is set when the axis speed is
REM at the velocity set by JOG VEL command.
INH 794   : REM wait until axis reaches desired speed
DWL 5   : REM wait 5 seconds before stopping
AXIS0 JOG OFF   : REM stop jogging
END

ENDP
```

Example 3

```
#DEFINE JogActiveX BIT792
#DEFINE JogAtSpeedX BIT794
#DEFINE StartProcess BIT3
#DEFINE inStopSpin BIT4
#DEFINE outJoggingX BIT32
#DEFINE outDispense BIT33
#DEFINE SysClock P6916
#DEFINE pDispenseTime P10
#DEFINE Message $V0
#DEFINE DriveEnableState BIT8476

PROG1
PROGRAM
_procINIT
REM wait for operator to press and release start input
INH StartProcess : INH -StartProcess
pDispenseTime=5.00
AXIS0 JOG ACC 500
AXIS0 JOG VEL 50
AXIS0 JOG DEC 500

_procEnableDrive
AXIS0 DRIVE ON
       DWL 0.2
IF (DriveEnableState=0)
       GOTO procError
       ENDIF

_procSPINNER
REM wait for operator to press and release start input
INH StartProcess : INH -StartProcess
AXIS0 JOG FWD
SET outJoggingX
INH JogAtSpeedX   : REM wait until axis reaches desired speed
SET outDispense   : REM turn on output to dispense fluid
SysClock=0   : REM use the system clock as a timer

WHILE (SysClock<pDispenseTime)
       IF (inStopSpin) BREAK
WEND

CLR outDispense
AXIS0 JOG OFF
INH -JogActiveX
CLR ourJoggingX
AXIS0 DRIVE OFF   : REM disable drive to allow for part change
GOTO procEnableDrive

_procError
       PRINT "Drive did not enable, check connections"
       AXIS0 DRIVE RES   : REM reset drive to clear errors
       DWL 3   : REM wait for reset to complete
       GOTO procEnableDrive
ENDP
```

JOG REN Transfer Current Position Into Jog Offset

Format	JOG REN {*axis* {*offset*}} {*axis* {*offset*}} ...
Group	Setpoint Control
Units	Units based on PPU
Data Type	FP32
Default	0
Prompt Level	PROGx
See Also	AXIS, BKL, BSC, CAM, GEAR, HDW, JLM, JOG, PPU
Related Topics	N/A
Product Revision	Command and Firmware Release

This command either clears or preloads the current position of a given axis and adds the difference to the jog offset parameter. The default *offset* argument is zero. The current position and jog offset are adjusted according to the following formulas:

jog offset = jog offset + current position - offset

current position = offset

If the optional *offset* argument is left out, it is set to zero. Otherwise, before the jog mode begins, the jog offset is reset as described in the **JOG RES** command.

Example

The following example transfers the X axis current position into the jog offset:

```
JOG REN X
```

JOG RES Transfer Jog Offset Into Current Position

Format	JOG RES {*axis* {*offset*}} {*axis* {*offset*}} ...
Group	Setpoint Control
Units	Units based on ppu
Data Type	FP32
Default	0
Prompt Level	PROGx
See Also	AXIS, BKL, BSC, CAM, GEAR, HDW, JLM, JOG, PPU
Related Topics	Axis Parameters (0-7) (8-15)
Product Revision	Command and Firmware Release

This command either clears or preloads the jog offset of a given axis and adds the difference to the current position. The default *offset* argument is zero. The current position and jog offset are adjusted according to the following formulas:

current position = current position + jog offset - offset

jog offset = offset

Example

The following example transfers the X axis jog offset into the current position:

```
JOG RES X
```

JOG REV Jog Axis Backward

Format	JOG REV {*axis*} {*axis*} ...
Group	Setpoint Control
Units	N/a
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	AXIS, BKL, BSC, CAM, GEAR, HDW, JLM, JOG
Related Topics	N/A
Product Revision	Command and Firmware Release

This command initiates a ramp to the velocity programmed with the **JOG VEL** command in the negative direction.

Example

The following example starts the Z axis jogging in the negative direction:

```
JOG REV Z
```

JOG SRC Set External Timebase

Format	JOG SRC *axis sourcedef* {*axis sourcedef*} ...
Group	Setpoint Control
Units	none
Data Type	N/A
Default	Clock
Prompt Level	PROGx
See Also	AXIS, SRC
Related Topics	N/A
Product Revision	Command and Firmware Release

This command specifies the timebase for jogging. See the SRC command for the definition of the *sourcedef* argument.

During each servo interrupt, the change in source pulses is multiplied by the servo period and the resulting delta time is fed into the jog mechanism. By default, jog is sourced off the CLOCK, feeding a single time unit per interrupt. Redirecting the jog source allows an external timebase to be used.

Example

The following example sets the X axis jog source to encoder 3:

```
JOG SRC X ENC3
```

Start motion:

```
JOG INC Z 0.10
```

JOG VEL Set Jog Velocity

Format	JOG VEL {*axis* {veloc}} {*axis* {veloc}} ...
Group	Setpoint Control
Units	units/second based on ppu
Data Type	FP32
Default	0
Prompt Level	PROGx
See Also	AXIS, BKL, BSC, CAM, GEAR, HDW, JLM, JOG, PPU
Related Topics	Axis Parameters (0-7) (8-15)
Product Revision	Command and Firmware Release

This command sets the programmed jog velocity for an axis. Issuing a **JOG VEL** command to an axis without an argument will display the current setting for that axis. The default jog velocity is 0.0 for all axes, therefore this command must be issued before any jogging can occur.

Example

The following example sets the X axis jog velocity to 10000 units/second:

```
JOG VEL X10000
```

JRK Set Jerk Parameter (S-curve)

Format	JRK {*rate*}		
Group	Velocity Profile		
Units	units/second^3 based on PPU		
Data Type	FP32		
Default	0		
Prompt Level	PROGx		
Report Back	Yes	**To view, type**	**JRK**
See Also	ACC, DEC, FVEL, IVEL, MASTER, PPU, STP, VEL		
Related Topics	Master Parameters (0-7) (8-15)		
Product Revision	Command and Firmware Release		

Description

Controls the slope of the acceleration versus time profile for coordinated motion.

Arguments

rate

Optional. Sets the slope for the master profile.

Remarks

The shifts between ramp and velocity can stress the mechanics of an application. In trapezoidal motion, the jerk (rate of change) in the acceleration and deceleration portions of a motion profile is at maximum. The change is nearly instantaneous.

The **JRK** command lets you introduce S-curve profiling. An S-curve provides a steady transition into the **ACC**, **DEC**, and **STP** rates. Reducing jerk decreases the strain on mechanics and improves position tracking. You can categorize three types of S-curve motion:

Standard Motion: For trapezoidal motion, the rate is zero. The corresponding acceleration profile is rectangular, indicating the maximum rate of change.

S-curve Motion: For S-curve motion, the rate is greater than zero. The corresponding acceleration profile is trapezoidal. The jerk rate helps ease the transition from one velocity to another. Looking at a velocity graph with **JRK** in use, the top and bottom of the **ACC**, **DEC**, and **STP** rates are smoothed.

Pure S-curve Motion: For pure S-curve motion, the rate is determined from an average acceleration rate you want to maintain. From the average acceleration rate, the actual accelerating is determined.

ACC= 2(average acceleration)

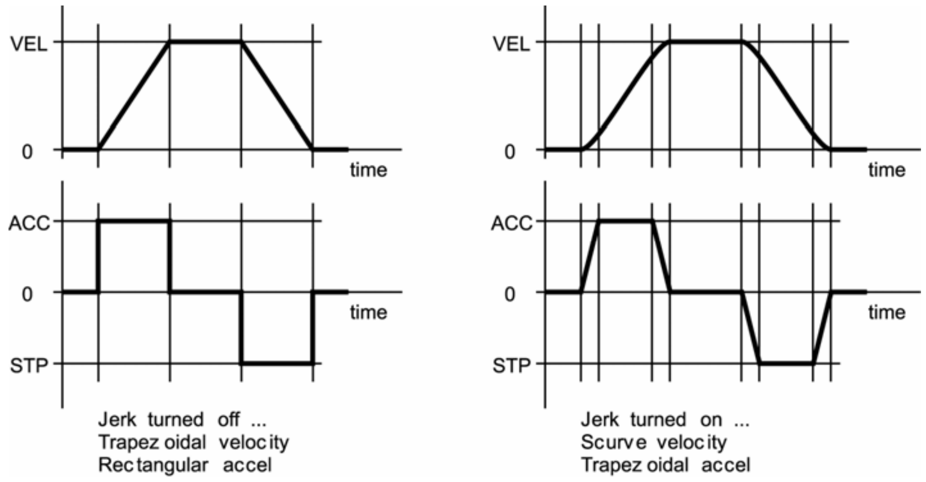
You can then give the jerk rate as an expression of acceleration and velocity.

JRK=(**ACC**^{2}/**VEL**)

Using an expression for the rate provides several advantages: it makes your code more readable, allowing you to quickly determine how the rate is derived; it increases accuracy because the DSP is able to take advantage of floating point numbers; and it avoids potential floating point rounding errors. If you change the **ACC** or **VEL**, the rate can adjust accordingly.

NOTE: We recommend using pure S-curve, providing the optimal acceleration profile with the least amount of mechanical stress.

The following illustrates the result of using s-curve jerk on a normal move:



Example

The following example sets the jerk ramp to 80000 units/second^3:

```
JRK ACC**2/VEL
```

KVF Feed Forward Gain for Position Velocity Loop

Format	KVF {*axis* {*value*}} {*axis* {*value*}} …
Group	Servo Control
Units	None
Data Type	FP32
Default	0
Prompt Level	PROGx
See Also	KVI, KVP
Related Topics	Axis Parameters (0-7) (8-15)
Product Revision	Command and Firmware Release

This command modifies the feed forward gain of position–velocity loop. The default value is zero, which should be typically set to a non-zero value before turning the PV (position-velocity) loop ON by **KVP**. Issuing a **KVF** command to an axis without an argument will display the current setting for that axis.

Example

The following example sets the X axis **KVF** to a value of 1.1.

```
KVF X 1.1
```

KVI Velocity Integral Gain for Position Velocity Loop

Format	KVI {*axis* {*value*}} {*axis* {*value*}} ...
Group	Servo Control
Units	None
Data Type	FP32
Default	0
Prompt Level	PROGx
See Also	KVF, KVP
Related Topics	Axis Parameters (0-7) (8-15)
Product Revision	Command and Firmware Release

This command modifies the Integral gain used in the position-velocity loop. Issuing a **KVI** command to an axis without an argument will display the current setting for that axis. The default value is 0.

Example

The following example sets the X axis **KVI** gain to 100.

```
KVI X 100
```

KVP Position Gain for Position Velocity Loop

Format	KVP {*axis* {*value*}} {*axis* {*value*}} ...
Group	Servo Control
Units	None
Data Type	FP32
Default	0
Prompt Level	PROGx
See Also	KVF, KVI
Related Topics	Axis Parameters (0-7) (8-15)
Product Revision	Command and Firmware Release

This command modifies the position gain in the position-velocity loop. Issuing a **KVP** command to an axis without an argument will display the current setting for that axis. The default value is 0, which implies that the PV loop is bypassed. Setting it to a non-zero value will turn on the PV loop.

Example

The following example sets the X axis **KVP** gain to 1.

```
KVP X 1
```

LIMIT Frequency Limiter

Format	LIMIT *index command* {*data*}
Group	Global Objects
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx
See Also	SRC, GEAR, CAM, RATCH
Related Topics	N/A
Product Revision	Command and Firmware Release

This command is used along with a second command to setup frequency limiters. The limiter *index* is a number from 0 to 7. Frequency limiters are sources that can limit the frequency of incoming pulses and redistribute large impulses over time.

Incoming pulses are multiplied by the limiter "multiplier" and accumulated over the limiter frame "width". At the end of each frame, the accumulated pulses are compared to the limiter "frequency" times the limiter "width" and any excess pulses are thrown away. The remaining pulses are redistributed evenly during the following frame.

The following is a list of valid limiter command combinations:

LIMIT SRC: Define limiter source.

LIMIT FREQ: Set frequency limit.

LIMIT WIDTH: Set pulse redistribution width.

LIMIT MULT: Set incoming pulse multiplier.

LIMIT FREQ Define Frequency Limit

Format	LIMIT *index* FREQ *frequency*
Group	Global Objects
Units	pulses / second
Data Type	FP32
Default	0
Prompt Level	SYS, PROGx
See Also	LIMIT
Related Topics	N/A
Product Revision	Command and Firmware Release

This command sets the limiter "frequency". The limiter "frequency" sets the maximum frequency allowed to pass through the limiter. The limiter "frequency" times the limiter "width" determine the maximum pulses per frame.

Setting limiter "frequency" to zero turns off the limiter's frame clipping and all pulses accumulated in the previous frame are redistributed over the next frame.

The *frequency* argument is a 32-bit floating point. Issuing a **LIMIT FREQ** command without an argument will display the current setting. The default frequency limit is zero.

Example

The following example sets the frequency limit of **LIMIT1** to 10000 pulses / second:

```
LIMIT1 FREQ 10000
```

LIMIT MULT Set Incoming Pulse Multiplier

Format	LIMIT *index* MULT *multiplier*
Group	Global Objects
Units	none
Data Type	FP32
Default	1
Prompt Level	SYS, PROGx
See Also	LIMIT, MULT
Related Topics	N/A
Product Revision	Command and Firmware Release

This command sets the limiter "multiplier". Incoming pulses are scaled by the "multiplier" before being accumulated into the frame buffer.

The *multiplier* argument is a 32-bit long integer. Issuing a **LIMIT MULT** command without an argument will display the current setting. The default pulse multiplier is one.

Example

The following example sets the pulse multiplier of **LIMIT3** to 100 times:

```
LIMIT3 MULT 100
```

LIMIT SRC Define Limit Source

Format	LIMIT *index* SRC *sourcedef*
Group	Global Objects
Units	None
Data Type	N/A
Default	None
Prompt Level	SYS, PROGx
See Also	LIMIT, SRC
Related Topics	N/A
Product Revision	Command and Firmware Release

This command sets the input source for a limiter. The default limiter source is NONE. See the SRC command for the definition of the *sourcedef* argument.

Example

The following example sets the source of **LIMIT0** to **RATCH2**:

```
LIMIT0 SRC RATCH2
```

LIMIT WIDTH Set Pulse Redistribution Width

Format	LIMIT *index* WIDTH *width*
Group	Global Objects
Units	Seconds/frame
Data Type	FP32
Default	0
Prompt Level	SYS, PROGx
See Also	LIMIT
Related Topics	N/A
Product Revision	Command and Firmware Release

This command sets the limiter "width". The limiter "width" sets the width of the limiter frame. The limiter "frequency" times the limiter "width" determine the maximum pulses per frame. Pulses from one frame are redistributed over the next frame.

Setting limiter "width" to zero causes the limiter to send multiplied pulses directly down the source chain without frame buffering. Setting the limiter "width" to too large of a value will cause unacceptable sluggishness in the limiter's response.

The *width* argument is a 32-bit floating point. Issuing a **LIMIT WIDTH** command without an argument will display the current setting. The default redistribution width is zero.

Example

The following example sets the pulse redistribution width of **LIMIT** number 2 to 50 milliseconds:

```
LIMIT2 WIDTH 0.050
```

LIST — List a Stored Program

Format	LIST {*first*} , {*last*}}
Group	Program Control
Units	None
Data Type	N/A
Default	N/A
Prompt Level	PLCx, PROGx
See Also	END, ENDP, PROGRAM
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Lists the currently selected program.

Arguments

first	First line number (or only line number) to be listed
last	Last line number to be listed

Remarks

The **LIST** command cannot be issued from within a program or while at the system level.

The arguments *first* and *last* define the listing range as follows:

LIST *first*	Lists a single line.
LIST *first*, *last*	Lists from "first" to "last".
LIST *first*,	Lists from "first" to end of program.
LIST ,*last*	Lists form start of program to "last".
LIST	Lists entire program.

NOTE: When using the **LIST** command to view a lineless program, you can view the automatic line numbers by setting bit 5651. (**SET 5651**)

Examples

Example 1

Lists entire program.

```
P00>LIST
PROGRAM
PBOOT : REM program will execute when controller power is turned on
_profile1
PRINT "RUNNING PROFILE"
X200
X100
X300
X400
X0
```

```
INH -516 : REM wait until motion is complete
PRINT "PROGRAM COMPLETE" : END
ENDP
```

Example 2

Lists entire program with line numbers.

```
P00>SET 5651
P00>LIST
10 PROGRAM
20 PBOOT : REM program will execute when controller power is turned on
30 _profile1
40 PRINT "RUNNING PROFILE"
50 X200
60 X100
70 X300
80 X400
90 X0
100 INH -516 : REM wait until motion is complete
110 PRINT "PROGRAM COMPLETE" : END
120 ENDP
```

Example 3

Lists a single line.

```
P00>LIST 40
40 PRINT "RUNNING PROFILE"
```

Example 4

Lists a range of lines from the program.

```
P00>LIST 50,80
50 X200
60 X100
70 X300
80 X400
```

Example 5

Lists up to line 50.

```
P00>LIST ,50
10 PROGRAM
20 PBOOT : REM program will execute when controller power is turned on
30 _profile1
40 PRINT "RUNNING PROFILE"
50 X200
```

Example 6

Lists from line 50 on.

```
P00>LIST 50,
50 X200
60 X100
70 X300
80 X400
90 X0
100 INH -516 : REM wait until motion is complete
110 PRINT "PROGRAM COMPLETE" : END
120 ENDP
```

LISTEN Listen to Program Output

Format	LISTEN
Group	Program Control
Units	None
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	LRUN, HALT, TRON, TROFF
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Links the current communication channel into a program's output.

Remarks

The **LISTEN** command cannot be issued from inside a program.

Normally when a program is run, the communication channel returns to the command prompt, allowing more commands to be entered. While at the command prompt, output from programs (including error reporting) is shut down to prevent mixing of command input and program output.

Issuing a **LISTEN** command suspends the command prompt—allowing program output to be monitored—until an escape character (ASCII 27) is received or the program ends.

The LRUN command runs a program while leaving the channel in the "listen" state.

Only a single program can be monitored by **LISTEN**/**LRUN** at a time.

Example

From the Terminal

The following issues a **LISTEN** command at the program 0 prompt. Program output is listed until "27" is entered, and the command prompt returns.

```
P00>LISTEN
(program output)
27
P00>
```

LOCK Lock Gantry Axis

Format	LOCK {*axis1 axis2*} {*axis1 axis2*} ...
Group	Setpoint Control
Units	none
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	AXIS, BSC, CAM, GEAR, HDW, JOG, UNLOCK
Related Topics	N/A
Product Revision	Command and Firmware Release

This command redirects "axis1" to follow the primary setpoint of "axis2". The actual position parameter of the axis is adjusted such that there is no change in following error when the primary setpoint switches. The **UNLOCK** command can be used to release the redirection. The default state of an axis is to follow its own setpoint.

Each axis generates a primary setpoint based on its current position, gear offset, jog offset, and cam offset. This number is normally used to tell the axis where it should be at any given time. The **LOCK** command tells an axis to use the primary setpoint of a different axis instead of its own. The **UNLOCK** command tells an axis to use its own primary setpoint once again.

You can only couple two axes together. For example, an eight axis controller can have four pairs of coupled axes.

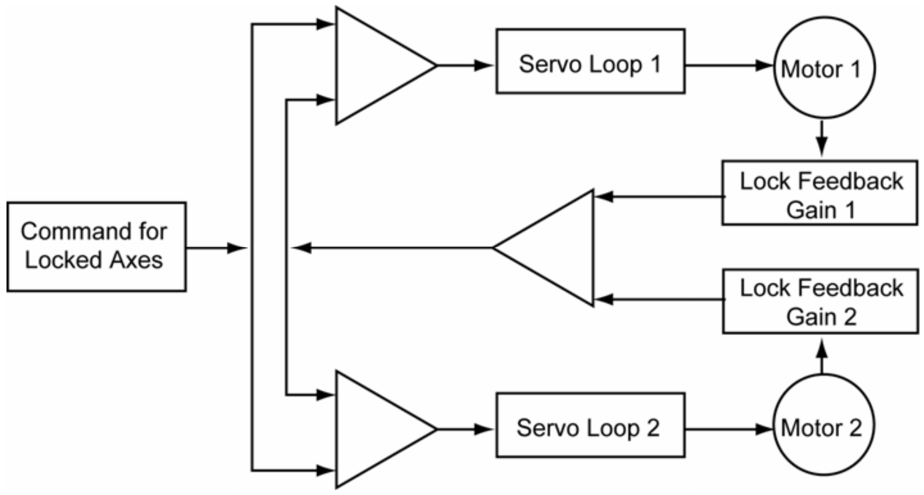
Example

The following example locks axis XB to the primary setpoint of axis XA:

```
LOCK XB XA
```

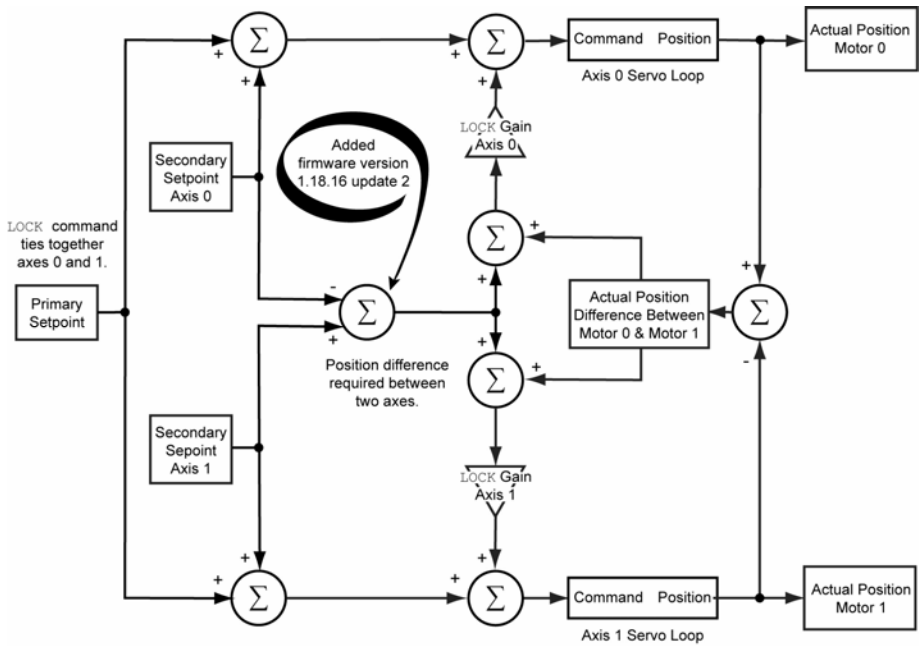
Lock Feedback Gain (Version 1.18.04)

The diagram below illustrates how the controller maintains the lock between axes.



Lock Feedback Gain (Version 1.18.16 update 2)

The diagram below provides a more detailed look at the **LOCK** command. With firmware version 1.18.16 update 2, the algorithm includes the commanded-position difference between axes, providing increased accuracy of the locked axes. It is


LOCK command ties together axes 0 and 1.
Primary Setpoint
Secondary Setpoint Axis 0
Secondary Sepoint Axis 1
Added firmware version 1.18.16 update 2
Position difference required between two axes.
LOCK Gain Axis 0
LOCK Gain Axis 1
Command Position
Axis 0 Servo Loop
Axis 1 Servo Loop
Actual Position Motor 0
Actual Position Motor 1
Actual Position Difference Between Motor 0 & Motor 1
+
-
Σ

Lock gantry axis

When two axes are locked together by using the **LOCK** command, then their primary setpoints become the same. In other words, the two axes will get exactly the same command signal. However, in the real world, the response of the two physical motors/actuators will be slightly different. To compensate for this error the user can turn on a feedback loop by setting some gain values for "Lock Feed Back Gain" parameter of the locked axes. The default value is zero, which forces this feedback loop to be off.

Example

```
P12376 = 3.5     : REM set lock gain axis 0
P12632 = 3.5     : REM set lock gain axis 1
LOCK Y X     : REM lock axis Y to axis X's primary setpoint
X /20    : REM start motion axis X, axis Y also moves due to lock
UNLOCK Y     : REM unlock axis Y
```

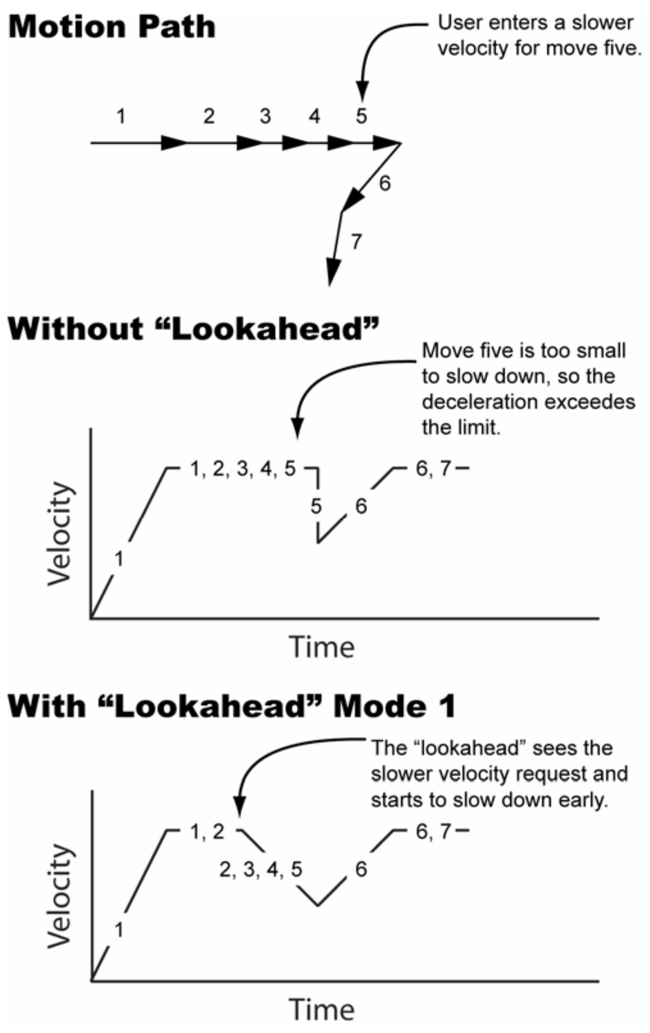
LOOK Look Ahead

Format	LOOK *command* {*data*}
Group	Velocity Profile
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	ACC, DEC, FVEL, IVEL, MASTER, MBUF, MOV, STP, VEL
Related Topics	Quaternary Master Flags (0-7) (8-15), Master Parameters (0-7) (8-15)
Product Revision	Command and Firmware Release

This special feature is used for velocity profiling. It acts as an intelligent observer and monitors and controls the velocity depending on the motion path shape and distance to travel.

You can use this command with or without the multi-buffer mode (**MBUF ON**). However it will be more effective with multi-buffer mode as there will be more moves buffered to look ahead, especially in case of tiny moves.

The following illustrates the use of Look Ahead mode:


Motion Path
User enters a slower velocity for move five.
1
2
3
4
5
6
7
Without "Lookahead"
Move five is too small to slow down, so the deceleration exceedes the limit.
1, 2, 3, 4, 5
5
6
6, 7
1
Velocity
Time
With "Lookahead" Mode 1
The "lookahead" sees the slower velocity request and starts to slow down early.
1, 2
2, 3, 4, 5
6
6, 7
1
Velocity
Time

LOOK ANG Set the Angles for Corner Sharpness

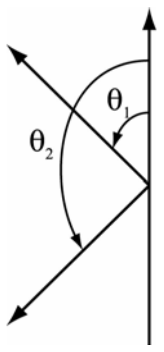
Format	LOOK ANG {*min_angle*, *max_angle*}
Group	Velocity Profile
Units	Degrees
Data Type	FP32
Default	Minimum angle= 0°; Maximum angle= 180°
Prompt Level	PROGx
See Also	ACC, DEC, FVEL, IVEL, MASTER, MBUF, MOV, STP, VEL
Related Topics	N/A
Product Revision	Command and Firmware Release

The Look ahead mode 1 uses the min angle (θ_1), max angle (θ_2) and sharpness of the corner to determine the velocity. The default value for θ_1 is 0 degrees; and θ_2 is 180 degrees.

For linear moves, the relationship between the angles and velocity profiler is as follows:

- Vector Turn Angle < θ_1: No change in velocity.
- Vector Turn Angle > θ_2: The velocity at the corner goes down to the master parameter "Lookahead Minimum Velocity".
- θ_1 < Vector Turn Angle < θ_2: The velocity goes proportionally from max to min as the turn angle goes from θ_1 to θ_2.

$$V_f = V^* \left(\frac{\theta_2 - \Psi}{\theta_2 - \theta_1} \right)$$



Where

V_f = Final velocity or sharp corner speed

V = **VEL**-User Programmed Velocity

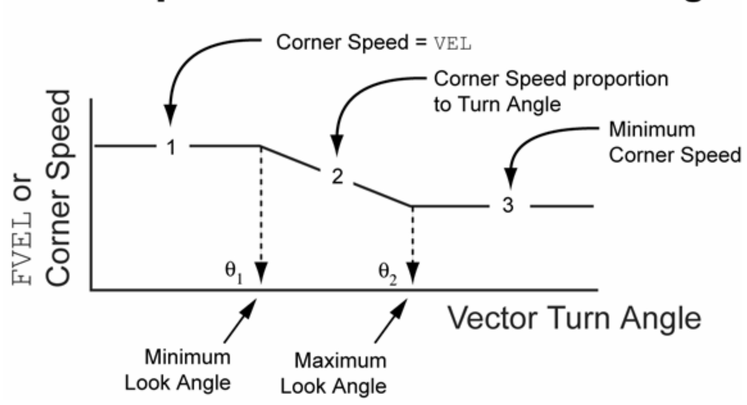
θ_1 = Minimum Look Angle

θ_2 = Maximum Look Angle

ψ = Vector Turn Angle

In case of a circular or arc moves the above is not used. Instead centripetal acceleration is used to calculate the master velocity as follows:

Corner speed as a Function of Turn Angle



Where

v = Velocity

a = Acceleration

R = Radius

Example

```
LOOK ANG (10, 90)
```

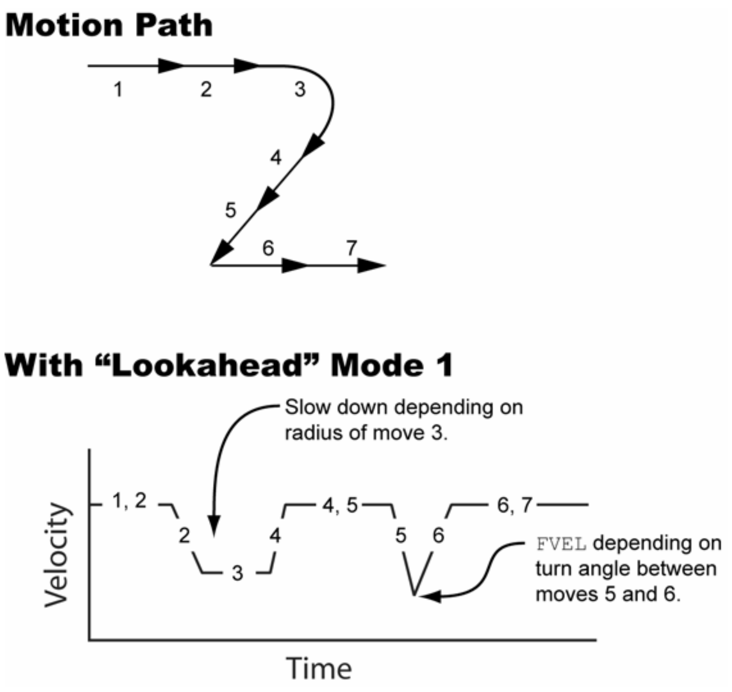
LOOK MODE Set Look Ahead Mode

Format	LOOK MODE {*number*}
Group	Velocity Profile
Units	None
Data Type	N/A
Default	0
Prompt Level	PROGx
See Also	ACC, DEC, FVEL, IVEL, MASTER, MBUF, MOV, STP, VEL
Related Topics	N/A
Product Revision	Command and Firmware Release

The default mode is 0. The mode 0 will work with any number of dimensions whereas mode 1 is only valid up to 3-dimensions.

- **Mode 0:** The controller tries to follow the user set velocities. It can see that the user has programmed a slower velocity at the end of so many moves and start to slow down in advance when there is not enough distance left.
- **Mode 1:** In addition to the above feature, this mode also looks at the geometry of the motion path. By doing so it gets the ability to foresee *sharp corners* and *small radius arcs* and automatically reduce speed according to the user set specifications. This mode is valid for up to 3 dimensions.

The following illustrates the use of Look Ahead mode 1:


Motion Path
1
2
3
4
5
6
7
With "Lookahead" Mode 1
Slow down depending on radius of move 3.
Velocity
1, 2
2
3
4
4, 5
5
6
6, 7
FVEL depending on turn angle between moves 5 and 6.
Time

Example

```
LOOK MODE 1
```

LOOK OFF Look Ahead Mode is Turned Off

Format	LOOK OFF
Group	Velocity Profile
Units	N/A
Data Type	N/A
Default	OFF
Prompt Level	PROGx
See Also	ACC, DEC, FVEL, IVEL, MASTER, MBUF, MOV, STP, VEL
Related Topics	N/A
Product Revision	Command and Firmware Release

The **LOOK OFF** command is used to turn off the lookahead mode for a master.

Example

```
LOOK OFF
```

LOOK ON Look Ahead Mode is Turned On

Format	LOOK ON
Group	Velocity Profile
Units	None
Data Type	N/A
Default	OFF
Prompt Level	PROGx
See Also	ACC, DEC, FVEL, IVEL, MASTER, MBUF, MOV, STP, VEL
Related Topics	N/A
Product Revision	Command and Firmware Release

The **LOOK ON** command is used to turn on the Lookahead feature for a particular master. Once this mode is turned on it will stay on unless the user explicitly turns it off. Issuing this command without an argument will display the current setting of this mode.

Example

```
P00> LOOK ON  : REM enable lookahead feature
P00>LOOK  : REM query look setting
LOOK ON
LOOK ANG (0, 180)
LOOK MODE 0
```

LOPASS Setup Lopass Filter

Format	LOPASS {*axis cutoff*} {*axis cutoff*} ...
Group	Servo Control
Units	Hertz
Data Type	FP32
Default	0
Prompt Level	PROGx
See Also	AXIS, DGAIN, FFVEL, FFACC, IGAIN, NOTCH, PGAIN,
Related Topics	Axis Parameters (0-7) (8-15)
Product Revision	Command and Firmware Release

This command initializes the second half of the output filter to act as a lopass filter, reducing high-frequency noise that may occur in a system. Setting the cutoff frequency to zero turns off the lopass filter.

Example

The following example sets the X axis lopass filter to a cutoff frequency of 500 hertz.

```
LOPASS X500
```

LRUN Run and Listen to a Program

Format	LRUN {*line*}
Group	Program Control
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	HALT, LISTEN, RUN, TROFF, TRON
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Runs the current program and leaves the communication channel linked to the program's output.

Arguments

line

Remarks

The **LRUN** command cannot be issued from inside a program.

Issuing an **LRUN** command with the optional *line* argument will start program execution at the given line number.

Normally when a program is run, the communication channel returns to the command prompt, allowing more commands to be entered. While at the command prompt, output from programs (including error reporting) is shut down to prevent mixing of command input and program output.

Issuing an **LRUN** command runs a program but does not return to the command prompt until an escape character (ASCII 27) is received or the program ends, allowing program output to be monitored.

The **LISTEN** command forces the communication channel back into this state from the command prompt.

Only a single program can be monitored by **LISTEN**/**LRUN** at a time.

NOTE: When using the **LIST** command to view a lineless program, you can view the automatic line numbers by setting bit 5651. (**SET 5651**)

Example

The following demonstrates how program communication functions when you send the **LRUN** command.

```
PROGRAM 0
SET 32
PRINT "bit 32 is set"
ENDP

P00>RUN : REM run program or PLC
P00>LRUN : REM List any print statements and run program
bit 32 is set
```

```
P00>LIST
 PROGRAM
_profile1
PRINT "RUNNING PROFILE"
 X200
 X100
 X300
 X400
 X0
 INH -516 : REM wait until motion is complete
 PRINT "PROGRAM COMPLETE" : END
 ENDP
P00>

P00>LRUN
RUNNING PROFILE
PROGRAM COMPLETE
P00>

P00>RUN
P00>LISTEN
PROGRAM COMPLETE
P00>
```

MASK Safe Bit Masking

Format	MASK *parameter*(*nandmask*, *ormask*)
Group	Logic Function
Units	N/A
Data Type	Long
Default	N/A
Prompt Level	SYS, PROGx
See Also	BIT, CLR, INH, SET
Related Topics	N/A
Product Revision	Command and Firmware Release

This command sets and clears multiple bits in a parameter and prevents the parameter from being corrupted by another program doing the same thing. The "*nandmask*" is used to clear bits and the "*ormask*" is used to set bits. The command replaces the following typical parametric expression:

parameter = (parameter AND NOT nandmask) OR ormask

NOTE: Do not use this command on any floating point parameter. Doing so causes an error—"Bad Data Type".

Example

The following example clears out the lower 8 bits of P4097 using 255 (FF hex) and replaces them with 85 (55 hex)

```
MASK P4097(255,85)
```

MASTER Direct Master Access

Format	MASTER *index command* {*data*}
Group	Global Objects
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx
See Also	AXIS, ENC, DAC
Related Topics	N/A
Product Revision	Command and Firmware Release

This command allows direct access to a master without having to be at the required program level. The master does not have to be attached to a program. The ***command*** argument can be any command from the velocity profile group.

Example

The following example sets the MASTER 2 velocity to 1000 and MASTER 4 feedrate override to 75 percent:

```
MASTER2 VEL 1000
MASTER4 FOV 0.75
```

MAXVEL Axis Velocity Limit

Format	MAXVEL {*axis*} {*value*}
Group	Axis Limits
Units	Units/second
Data Type	FP32
Default	0
Prompt Level	PROGx
See Also	AXIS, MOV, TMOV, VEL
Related Topics	Axis Parameters (0-7) (8-15), Secondary Master Flags (0-7) (8-15)
Product Revision	Command and Firmware Release

This command sets the velocity limit for individual axis. This is useful for optimizing the speed of the machine with axes that can handle different velocity limits. Depending on the axes involved in the move and the size of their moves, the profiler will automatically adjust to make a maximum velocity move, overriding the **VEL** value for the move. This mode can be used with the **TMOV** command, as well.

The maxvel is stored in the axis parameter "MaxVel" and its default value is zero. When all the axes attached to a master have the **MAXVEL** value set to greater then zero, this mode is automatically turned on. This is indicated by master secondary flag 'SlaveMaxVel'. This mode will turn off, if one or more of the attached axes **MAXVEL** velocities are set to zero or by clearing the master secondary flag 'Slave MaxVel'.

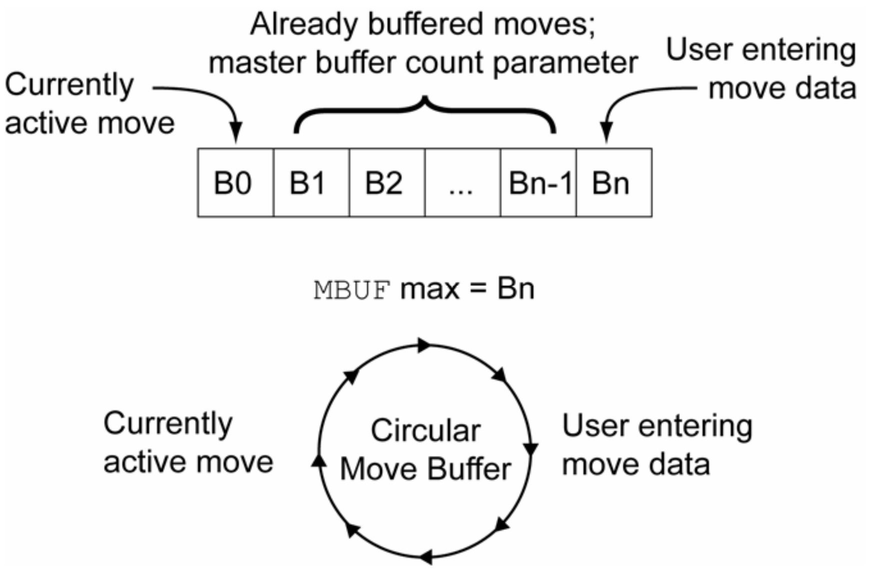
Example

```
MAXVEL X 5
MAXVEL Y 2
```

MBUF Multiple Move Buffer

Format	MBUF *command*
Group	Velocity Profile
Units	N/A
Data Type	N/A
Default	OFF
Prompt Level	PROGx
See Also	DIM, LOOK, MOV, TMOV
Related Topics	Quaternary Master Flags (0-7) (8-15), Master Parameters (0-7) (8-15)
Product Revision	Command and Firmware Release

This command is used along with a second command to define the length of the move buffer. The default value for the move buffer is 2 i.e., one active move and one buffered move. This default move buffer is in the system memory. In some applications the user may want to increase the number of moves buffered. This can be done by using **DIM MBUF** command from **PROG** level prompt to allocate program level user memory for the move buffer.



MBUF OFF Single Move Will be Buffered for the Motion Profiler

Format	MBUF OFF
Group	Velocity Profile
Units	None
Data Type	N/A
Default	OFF
Prompt Level	PROGx
See Also	DIM, LOOK, MOV, TMOV
Related Topics	Quaternary Master Flags (0-7) (8-15)
Product Revision	Command and Firmware Release

The **MBUF OFF** command is used to turn off the master multi-buffer mode. This command will wait for master in motion flag to clear and then it will clear all the buffers. By turning the mode off, the controller will go back to its default state, that is, one active move and one buffered move.

Example

```
MBUF OFF
```

MBUF ON Multiple Moves Buffered for Motion Profiler

Format	MBUF ON
Group	Velocity Profile
Units	None
Data Type	N/A
Default	OFF
Prompt Level	PROGx
See Also	DIM, LOOK, MOV, TMOV
Related Topics	Quaternary Master Flags (0-7) (8-15)
Product Revision	Command and Firmware Release

The **MBUF ON** command is used to set the master in multi-buffer mode. Issuing just **MBUF** command will display the current length of the buffer and its status, whether it is on or off.

By using this mode one can increase the throughput (moves per second), because the tiny moves can be buffered in advance and consequently cut the processing time. Besides this one can use features like Lookahead more effectively.

Example

```
CLEAR
DIM MBUF (20)
MBUF ON
:
:
MBUF OFF
```

NOTE: **DIM MBUF** command should be issued each time the **MBUF** is turned on.

MEM Display Memory Allocation

Format	MEM
Group	Memory Control
Units	Bytes
Data Type	Long
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	DIM, CLEAR
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Displays the amount of memory remaining, in bytes.

Remarks

The **MEM** command cannot be issued from within a program.

From the system level, the **MEM** command displays the amount of memory that can be allocated to a program.

From the program level, the command displays the amount of memory available for program, move buffer, variable, and array storage.

Example

```
MEM
```

MODE Binary Data Formatting

Format	MODE {*mode*}
Group	Operating System
Units	None
Data Type	N/A
Default	FIFO= 0; COMx= 1
Prompt Level	SYS, PROGx, PLCx
See Also	CLOSE, OPEN
Related Topics	N/A
Product Revision	Command and Firmware Release

This command controls the encoding and decoding of the data fields in immediate mode commands (see Binary Host Interface in the Programmer's Guide.) Issuing a **MODE** command without an argument displays the current setting. The default setting for the FIFO channel is 0 and the default for the COM1 and COM2 channels is 1.

Control character prefixing and high bit stripping follow Kermit communications protocol conventions. The escape code for control prefixing is the number sign (#), and the escape code for high bit stripping is the ampersand (&).

These sequences were added primarily for the serial communication channels. The control prefixing was added to prevent valid data within a binary packet from being confused with the XON / XOFF flow control codes. The high bit stripping was added for cases in which a 7-bit data path must be used. In general, the FIFO channel does not require these precautions.

The following table lists the valid data formatting modes. Note that it is not possible to activate high bit stripping without also activating the control character prefixing.

This table shows the data format modes:

Mode Value	High Bit Stripping	Control Prefixing
0	OFF	OFF
1	OFF	ON
2	OFF	OFF
3	ON	ON

Example

The following example turns on both control prefixing and high bit stripping:

```
MODE 3
```

MOV Define a Linear Move

Format	MOV {*axis target*} {*axis target*} ...
Group	Interpolation
Units	Target= units based on PPU
Data Type	FP32
Default	N/A
Prompt Level	PROGx
Report Back	None **To view, type** N/A
See Also	MASTER, MAXVEL, NURB, PPU, SINE, SPLINE, TMOV, TRJ
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Moves to the position using the coordinated motion profiler.

Arguments

axis

Optional. Assigns an axis.

target

Optional. Position at the end of the move (in units).

Remarks

It is not necessary to use the **MOV** command; you can provide the axis name and the *target* in a program, or at the terminal without the **MOV** command (see the example below).

When a slash mark (/) trails the *axis* argument, the move is interpreted as an incremental move for the number of units specified, rather then an absolute move to a position.

Moves Buffer

While the coordinated motion profiler executes a move statement, it executes subsequent code statements until it encounters another move statement. The profiler then buffers that move statement. When the current move statement completes, the profiler executes the buffered move statement and executes subsequent statements, and looks for the next move statement to buffer.

Examples

Example 1

Moves the X axis to the absolute position of 10 units.

```
ACC 750 DEC 500 VEL 75 STP 750
MOV X10
```

Example 2

The command **MOV** is not necessary. The following also moves the X axis to the absolute position of 10 units.

```
X10
```

Example 3

When you command motion for multiple axes on a single line, the controller treats it as coordinated motion. The X and Y axes complete their respective moves at the exact same time.

```
X20 Y-30
```

Example 4

To move an incremental distance, use a slash mark (/) following the axis. In this example, the X axis moves an incremental distance of 20 units from its current position. Then the Y axis moves a decremental distance of 30 units from its current position.

```
X/20
Y/-30
```

Example 5

The X axis makes an incremental move, Y axis makes an absolute move, and Z axis makes a decremental move. For this coordinated move, all axes complete their moves at the exact same time.

```
X/2 Y2 Z/-2
```

Example 6

A coordinated move with the X axis doing linear-interpolation and the Y axis doing sinusoidal interpolation.

```
X2 SINE Y(0,90,90,100)
```

MSEEK Marker Seek Operation

Format	MSEEK {*axis*(*incmove*, *mode*) {*capture_register*}} {*axis*(*incmove*, *mode*) {*capture_register*}} ...
Group	Feedback Control
Units	Incmove= units based on PPU
Data Type	Incmove= FP32
Default	Mode and Capture register see INTCAP
Prompt Level	PROGx
Report Back	None **To view, type** N/A
See Also	ACC, AXIS, INTCAP, MOV, PPU, STP, VEL
Related Topics	N/A
Product Revision	Command and Firmware Release

This command initiates a marker seek operation. A master can only control one **MSEEK** at a time. If multiple axes are indicated, they will execute in the order that they appear.

NOTE: Refer to the mode parameter and hardware capture register information in the **INTCAP** command section. The mode parameter and hardware capture register for the **MSEEK** is the same as those used in the **INTCAP** command.

A marker seek operation is as follows:

1. Start an incremental move. Start looking for marker.
2. When marker is located, decelerate to a stop.
3. Reverse direction and move back to where marker was located.
4. Reset encoder position to zero and terminate **MSEEK** mode.

If the incremental move ends without the marker being located, the corresponding "capture complete" flag will not be set, and the encoder position is not reset to 0. Typically, the incremental move should be large enough to guarantee a complete revolution (at least 1.5 revolutions are suggested.)

NOTE: The incremental move is specified in units, and all previous master settings for **ACC**, **STP**, and **VEL** are used as motion parameters.

When an **MSEEK** command is performed, the FLZ offset register is cleared by the processor. It is recommended that the user also clears the CAM, Gear, and Jog registers by performing the following command sequence:

```
CAM OFF

CAM RES

GEAR RES

JOG OFF

JOG RES
```

Example 1

The following example assumes **ENC0** as position feedback on **AXIS0 (X)**. The **MSEEK** command moves the X axis to its marker position.

```
MSEEK X(10000,0)
```

Example 2 (version 1.18)

The following example moves AXIS0 (X) until the fourth marker (axis one) turns on. This position of AXIS0 (X) is then captured in capture register 2.

```
MSEEK X(10000,9) CAP2
```

MULT Set Encoder Multipliers

Format	MULT {*axis* {*mode*}} {*axis* {*mode*}} …
Group	Feedback Control
Units	N/A
Data Type	Integer
Default	ACR9000 only= 4; All other= 1
Prompt Level	PROGx
See Also	AXIS, MOV, PPU
Related Topics	N/A
Product Revision	Command and Firmware Release

This command sets up count direction and hardware multiplication for the encoder attached to the given axis. Issuing the **MULT** command to an axis with no argument will display the current setting. The default setting for ACR9000 axes is 4; for all other controllers the default is 1.

Mode	Description
0	0x multiplier, encoder turned off, no quadrature counts
1	1x multiplier, count up on rising edge of A channel
2	2x multiplier, count up on both edges of A channel
4	4x multiplier, count up on edge of either channel
-1	1x multiplier, count down on rising edge of A channel
-2	2x multiplier, count down on both edges of A channel
-4	4x multiplier, count down on edge of either channel

Example

The following example sets hardware multiplication for axis X to 1 and axis Y to 2:

```
MULT X1 Y2
```

NEW Clear Out a Stored Program

Format	NEW {*command*}
Group	Program Control
Units	None
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	CLEAR, END, ENDP, ERASE, FLASH, HALT, PROGRAM
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Erases the currently selected program.

Arguments

command

Remarks

If you attempt to erase a program that is currently running, the controller generates an error.

The **NEW** command cannot be issued from within a program.

The following is a list of valid command combinations:

NEW ALL	Erases all user and PLC programs not currently running.
NEW PLC	Erases the corresponding PLC program.
NEW PROG	Erases the corresponding user program.

NOTE: When you erase programs, the data is unrecoverable.

Example

```
NEW PROG1 : REM CLEAR OUT PROGRAM 1
```

NEW ALL Clear Out a Stored Program

Format	NEW ALL
Group	Program Control
Units	None
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	CLEAR, END, ENDP, ERASE, FLASH, HALT, PROGRAM
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Erases all user and PLC programs that are not currently running.

Remarks

The **NEW** command cannot be issued from within a program.

NOTE: When you erase programs, the data is unrecoverable.

Example

```
NEW ALL : REM CLEAR OUT PROGRAMS
```

NEW PLC Clear Out a Stored Program

Format	NEW PLC *number*
Group	Program Control
Units	None
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx,
See Also	CLEAR, END, ENDP, ERASE, FLASH, HALT, PROGRAM
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Erases the corresponding PLC program.

Remarks

The **NEW** command cannot be issued from within a program.

If you attempt to erase a program that is currently running, the controller generates an error.

NOTE: When you erase programs, the data is unrecoverable.

Example

```
NEW PLC1 : REM CLEAR OUT PLC PROGRAM 1
```

NEW PROG Clear Out a Stored Program

Format	NEW PROG *number*
Group	Program Control
Units	None
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx,
See Also	CLEAR, END, ENDP, ERASE, FLASH, HALT, PROGRAM
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Erases the corresponding user program.

Remarks

The **NEW** command cannot be issued from within a program.

If you attempt to erase a program that is currently running, the controller generates an error.

Example

```
NEW PROG1 : REM CLEAR OUT PROGRAM 1
```

NORM Normalize Current Position

Format	NORM {*axis* {*length*}} {*axis* {*length*}} ...
Group	Feedback Control
Units	units
Data Type	FP32
Default	N/A
Prompt Level	PROGx
See Also	AXIS, PPU, REN, RES, ROTARY
Related Topics	N/A
Product Revision	Command and Firmware Release

This command normalizes the current position of an axis. A **MOD** operation is done on the current position, resulting in a new current position between zero and the *length* argument. The primary setpoint and the encoder count are adjusted accordingly in order to prevent the axis from jumping. If the *length* argument is left out, the rotary length set by the **ROTARY** command is used.

Example

The following example normalizes the A axis to 360 units:

```
NORM A360
```

NOTCH Setup Notch Filter

Format	NOTCH {*axis*(*center*, *width*)} {*axis*(*center*, *width*)} ...
Group	Servo Control
Units	Hertz
Data Type	FP32
Default	0, 0
Prompt Level	PROGx
See Also	AXIS, DGAIN, FFACC, FFVEL, IGAIN, LOPASS, PGAIN
Related Topics	Axis Parameters (0-7) (8-15)
Product Revision	Command and Firmware Release

This command sets up the first half of the output filter to act as a notch filter, reducing mechanical resonance that may occur in a system. Setting the center frequency to zero turns off the notch filter.

Example

The following example sets the X axis notch filter to a center frequency of 100 hertz and bandwidth of 50 hertz.

```
NOTCH X(100,50)
```

NURB Non-Uniform Rational B-Spline Interpolation

Format	NURB *command*
Group	Interpolation
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	MASTER, SPLINE
Related Topics	Tertiary Master Flags (0-7) (8-15), Master Parameters (0-7) (8-15)
Product Revision	Command and Firmware Release

NOTE: This command does not apply to the ACT1505 or ACR8000 controller.

With NURB (non-uniform, rational B-spline) interpolation, the NURB curve points generated by a CAD/CAM package can be directly downloaded to the controller. Thus, no need to generate and download huge amounts of data approximating the NURB curve with small linear moves. The CAD/CAM package creates the NURB data with tool compensation.

The following is a list of valid **NURB** command combinations:

NURB MODE: Enable NURB Interpolation Mode Type.

NURB RANK: Set NURB Rank value.

NURB END: End NURB Interpolation.

The following is a typical single **NURB** command format for a 2-D curve with X and Y axes.

```
K 3 X5 Y/2 W2.3 VEL 5
```

Data	Description
`K 3`	Knot Value of 3
`X5`	Absolute control point for x-axis
`Y / 2`	Incremental control point for y-axis
`W 2.3`	Weight of x-y control point
`VEL 5`	velocity from previous knot to this knot

The weight (*W*) and velocity (*VEL*) are optional in the above command, and if omitted, the previous value is used.

The NURB curve is defined using these variables:

R = Rank

N = Degree = R-1

P = Control point

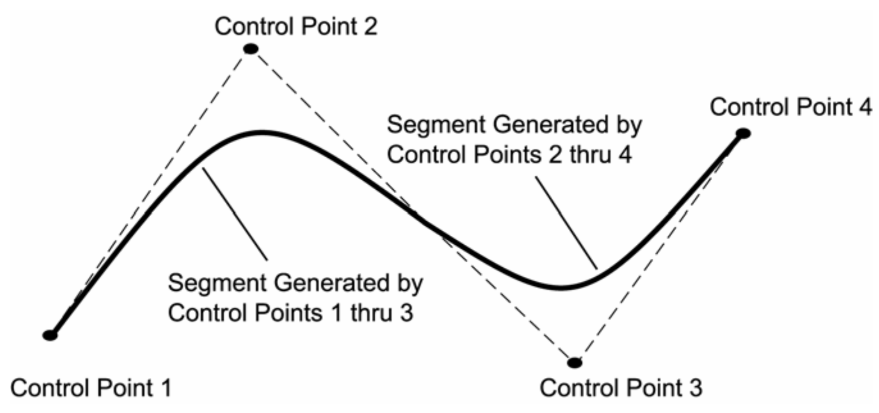
W = Weight of each control point; number of W = number of P

K = knot; number of knots number of P + R

u = NURB parameter

Rank

Rank is used to define how many control points are used to generate an individual segment within a nurb curved line. The following illustration uses a **NURB RANK** value of Rank 3 (degree2). There are two segments that generate the **NURB** curved line using four (4) control points.


Control Point 2
Segment Generated by Control Points 2 thru 4
Control Point 4
Segment Generated by Control Points 1 thru 3
Control Point 1
Control Point 3

Weight

The default value is “1” and the keyword ‘W’ is used to define the weight of a control point. Increasing the weight of a control point will result in increasing the effect of control point on the **NURB** curve and vice versa.

Knot

Keyword ‘K’ is used to define the knots of NURB curve.

Number of knots = Number of Control Points + NURB Rank

The first and last R knots (where R is the rank) must be specified as duplicated values. In other words, the first R knots should be zero and the last R knots should have the same value.

The knots are a strictly non-decreasing function and giving a negative value to a knot (i.e.–1) will end the NURB move block.

Control Point

The control points are given as the target points to the axes. These could be absolute or incremental values.

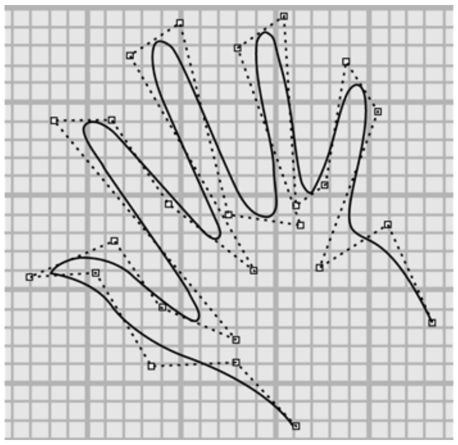
A NURB curve starts from the first control point and ends at the last control point. The first control point must be current position of axes or the target point of the previous move.

The following example uses the control points as indicated in the following figure to generate the resulting NURB curve as traced.

NOTE: In this mode, the controller is using N+R+1 points to calculate NURB curve. If stopped by **DWL** or **INH** commands, the move will stop R+1 points before the command appeared.

At the point where the negative knot is read by NURB profiler, the profiler knows that the block is ending, so the user should keep the speed and distance in the last segment such that there is enough time to slow down.

The following illustrates the use of NURB interpolation:



Example

```
PPU X 1000 PPU Y 1000
ACC 1 IVEL 0    : REM Initial velocity of zero is set
NURB RANK 4    : REM Rank of 4 (i.e. a degree of 3 is selected)
NURB MODE 1    : REM Dynamic Interpolation Mode is selected.
REM This command should be given each time before starting the
REM NURB block.
K 0 X23 Y8 VEL 4
K 0 X21 Y13.4 VEL 5
REM The first four multiple knots of zero VEL 5 are used from the
REM 1st to 2nd control point.
K 0 X17.4 Y11.2 VEL 4
REM VEL 4 will be used from 2nd to 5th control point.
K 0 X20.6 Y19.5
K 2.96 X18.8 Y22.1
K 5.83 X17.7 Y15.5 VEL 2
K 7.17 X16.1 Y14.5 VEL 4
K 9.82 X15.5 Y24.6 W2
REM A weight of 2 is used for this control point.
K 10.82 X13.1 Y22.8
K 13.76 X16.4 Y13.4
K 14.98 X12.5 Y14
K 18.1 X10 Y24.2 W1.1 VEL 5
REM VEL command used to control the speed.
K 19.58 X7.2 Y22.5
K 22.6 X13.9 Y11
K 24.04 X9.3 Y14.5
K 27.04 X6.3 Y19
K 28.34 X3.3 Y19
K 31.17 X12.9 Y7.2
K 32.58 X9.03 Y8.9
K 36.3 X6.4 Y12.6
K 37.79 X1.9 Y10.6
K 40.39 X5.5 Y10.8
K 41.92 X8.4 Y5.8
K 43.98 X13.1 Y6.1
K 46.21 X16.8 Y2.7 VEL 0.2    : REM Finally slowing down.
K 49.4    : REM The last four multiple knots.
K 49.4
K 49.4
K 49.4
K -1    : REM Negative knot indicates that NURB block has ended.
```

NURB END Ends Nurb Interpolation Mode

Format	NURB END
Group	Interpolation
Units	None
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	MASTER, SPLINE
Related Topics	Tertiary Master Flags (0-7) (8-15)
Product Revision	Command and Firmware Release

This command is used to terminate the NURB interpolation mode initiated by the **NURB MODE** command.

The NURB ending is automatically done, if the NURB motion has normally come to end/stop by a negative knot. However, if stopped abnormally, like issuing incomplete or wrong data to the NURB profiler, then this command must be used to terminate the NURB mode.

Example

```
NURB END
```

NURB MODE Enable NURB Mode

Format	NURB MODE {*value*} ...
Group	Interpolation
Units	N/A
Data Type	Integer
Default	0
Prompt Level	PROGx
See Also	MASTER, SPLINE
Related Topics	Tertiary Master Flags (0-7) (8-15)
Product Revision	Command and Firmware Release

A **NURB MODE** command must be issued each time before a NURB block of moves. The subsequent move commands are treated as NURB control points, and NURB interpolation is used to trace the NURB curve. This modes remains active until a negative knot is received, Kill Move Flag is set, or the user issues a **NURB END** command.

NURB Interpolation Modes

To give control over machine speed the following modes are available. The user decides which one to use, depending upon the application.

Mode	NURB Type	TOV	FOV	Description
0	Time Interpolation	Yes	NA	The knots are taken as time. Automatically adjusts velocity and acceleration depending on the knots. The master parameter NURB Time Factor may be used to scale the knots to time and change the speed. Very smooth and good for high speed precise moves.
1	Dynamic Interpolation	Yes	Yes	User specifies the speed and acceleration limits. Depending on the curvature of the curve and acceleration/velocity limit set be the user, automatically controls the speed. User should watch for sharp turns and slow down in time to make a smooth turn.
2	Linear Interpolation	Yes	NS	Linearly maps linear-arc-length/VEL time to knots. The velocity command starts effective from the start of the segment.
3	NR Interpolation	Yes	Yes	Newton Raphson Approximation for Vector Velocity Control
4	Smooth Interpolation	Yes	NS	Uses Cubic Spline to map arc-length/VEL time to knots. The smoother trajectory comes at the cost of slowing down close to each knot to make a smoother transition.

NA: Not Applicable

NS: **FOV** value is applied to the next segment

Example

The following examples selects Time Interpolation Mode for all following **NURB** commands.

```
NURB MODE 0
```

NURB RANK Set the Order of NURB Interpolation

Format	NURB MODE {*value*} ...
Group	Interpolation
Units	N/A
Data Type	Integer
Default	4
Prompt Level	PROGx
See Also	MASTER, SPLINE
Related Topics	N/A
Product Revision	Command and Firmware Release

The default NURB rank is 4 (degree of 3). The valid values for NURB rank are 2,3,4,5. The user can change this value by issuing the **NURB RANK** command. This command should not be used while the NURB profiler is in motion

Example

The following example sets the NURB Rank to 3 (degree of 2).

```
NURB RANK 3
```

OFFSET Absolute Program Path Shift

Format	OFFSET {*axis* {*offset*}} {*axis* {*offset*}} ...
Group	Transformation
Units	Units based on PPU
Data Type	FP32
Default	0
Prompt Level	PROGx
See Also	FLZ, PPU, ROTATE, SCALE
Related Topics	N/A
Product Revision	Command and Firmware Release

This command will cause the programmed path to be shifted. The amount of the path shift is defined by the "offset " data. If the offset value for an axis is not specified, the zero location for that axis will be equal to its current location.

Example

The following example sets the pulses per unit to 8192 and an offset of 50. While axis X is commanded to position 0, its actual position is 50, and the target position is computed as 409600 instead of zero (409600/8192=50).

```
X0
PPU 8192
OFFSET X 50
X0
```

OOP Output On Position

		Product	Revision
Format	OOP *axis* {*command*(*data*)}		
Group	Feedback Control	ACR1505	1.18.15
Units		ACR8020	n/a
Data Type	See above	ACR9000	1.24
Default	See above	ACR9030	1.24
Prompt Level	SYS, PROGx	ACR9040	n/a
See Also	IHPOS, INT, INTCAP, MSEEK, OOP BASE, OOP OFF, PLS	Aries CE	n/a
Related Topics	N/A		

Description

This command provides the capability to generate outputs based on encoder, and is used alone or in conjunction with another (daughter) command (**BASE**, **INC**, **OFF**, or **RND**).

Remarks

When a compared encoder value equals the desired value, a triggered output pulse is generated by the ACR1505/9000/9030 Encoder FPGA hardware. There are two modes of operation for the high-speed outputs: Incremental (**OOP INC**) and Random (**OOP RND**).

Encoders 8-15 for ACR9000/9030 and Encoders 2-19 for ACR1505 are not supported by the OOP command.

In addition to the position-based output pulses, **OOP** can generate user-specified time-based pulses. These time-based pulses follow each position-based pulse. If the time-based components of the command are omitted, then the **OOP** command does not generate time-based outputs.

For the ACR9000/9030, the **ENCn SRC5** (where n is the axis number) can be employed to allow using the **OOP** feature with stepper axes. See the ENC SRC command for additional information on how to link the encoder parameter to the step count.

When **OOP** is used, the output control flag (such as Bit 32) no longer controls the physical output. To return control of the physical output to the appropriate control flag, issue `OOPn OFF` (for Bit 32, this would be `OOP0 OFF`).

When **OOP** is controlling the physical output, the output control flag (such as Bit 32) does not indicate the status of the physical output.

For the ACR9000/9030, the standard outputs (OUT32-OUT39) optical isolators prevent pulse widths narrower than 1.3 milliseconds. For pulse widths as low as 20 nanoseconds, the pulses from OOP0–OOP2 may be optionally steered to high speed outputs on the Axis 3 connector. For Encoders 0, 1, and 2, the physical output for the OOP pulse is defined by bit index 10 in the encoder flags of the corresponding encoder. For example, for OOP0, Bit 2570 steers the output. If the bit is not set, pulses are steered to the OUT Bit corresponding to that encoder. If the bit is set, pulses are steered to one of the high speed outputs associated with Encoder 3 as listed in the following:

- OOP0 – Axis 3 step output
- OOP1 – Axis 3 direction output
- OOP2 – Axis 3 SSI clock (Channel A Encoder automatically set as output)
- OOP3 – can only go to OUT35. Note that if the SSI clock is selected for OOP2, then Encoder 3 cannot be used for OOP because channel A is not available.
- OOP4 – Axis 7 step output
- OOP5 – Axis 7 direction output
- OOP6 – Axis 7 SSI clock (Channel A Encoder automatically set as output)
- OOP7 – can only go to OUT39. Note that if the SSI clock is selected for OOP5, then Encoder 7 cannot be used for OOP because Channel A is not available.

The OOPn OFF command is required to restore the output (such as OUTn, step, direction, or Channel A of ENC3 and ENC7) to its original function.

Arguments

axis	Specify the OOP encoder number (0-7 for ACR9000/9030, 0-1 for ACR1505)
command	Daughter command used in conjunction with OOP
data	Requirements are based upon the daughter command

Example

The following queries the current setup for OOP for Encoder 0.

```
P00>OOP 0
OOP 0 INC  ( 1, 0, 0, 0, 0, 1 )
OOP 0 OFF
```

OOP BASE Output On Position - Storage Array

		Product	Revision
Format	OOP *axis* BASE LA(*number*)		
Group	Feedback Control	ACR1505	1.18.15
Units	Microseconds	ACR8020	n/a
Data Type	Long Array	ACR9000	1.24
Default	Null	ACR9030	1.24
Prompt Level	SYS, PROGx	ACR9040	n/a
See Also	IHPOS, INT, INTCAP, MSEEK, OOP RND, PLS	Aries CE	n/a
Related Topics	Encoder Flags (0-7)		

Description

The **OOP BASE** command selects which storage array base to use for random count-data input.

Remarks

The base array is a 32-bit long integer array (LA). For the ACR1505, the array length must be a multiple of 4. For the ACR9000/9030, the array size must be at least two, but it is not constrained to multiples of 4.

Arguments

number

Examples

See OOP RND for a complete example.

OOP INC Output On Position - Incremental Mode

Format	OOP *axis* INC(*width*, *start*, *stop*, *interval*, {*time-based number*, *time-based period* })		
Group	Feedback Control	**Product**	**Revision**
Units		ACR1505	1.18.15
Data Type	See above	ACR8020	n/a
Default	See above	ACR9000	1.24
Prompt Level	SYS, PROGx	ACR9030	1.24
See Also	IHPOS, INT, INTCAP, MSEEK, OOP OFF, PLS	ACR9040	n/a
Related Topics	N/A	Aries CE	n/a

Description

The **OOP INC** command enables incremental mode. The mode is defined by a start position value (*Start*), an end position value (*Stop*), and an interval value (*interval*).

The **OOP** command generates the first pulse output at the start position and then repeats the pulse output at positions *Start* + *interval*, *Start* + *2 x interval*, *Start* + *3 x interval*, and so on. The interval may be positive or negative. The software terminates the **OOP** when the encoder count reaches position Stop.

When **OOP** is used, the output control flag (such as Bit 32) no longer controls the physical output. To return control of the physical output to the appropriate control flag, issue `OOPn OFF` (for Bit 32, this would be `OOP0 OFF`).

When **OOP** is controlling the physical output, the output control flag (such as Bit 32) does not indicate the status of the physical output.

Arguments

axis Specifies which axis output to use. For ACR1505, there are only two high-speed axes—Axis 0 and Axis 1. For the ACR9000/9030, each axis has an OOP which can be configured for either normal speed, 1.3 ms, or high speed, 0.20 µs (for OOP 0-2 and 4-6 only, not applicable to OOP 3 and 7).

mode Determines which mode to use. For incremental mode, type `INC`. For random mode, type `RND`.

width Sets the generated pulse width in microseconds. The ACR1505, the range is 0.033 – 1000 µs. For the ACR9000/9030, the allowable pulse widths are 0.020 to 40000000 µs, with a resolution of 0.020 µs.

start Sets the position for the first pulse. The position is in encoder counts or steps.

stop Sets the position for the last pulse. The position is in encoder counts or steps.

interval
Sets the encoder-count interval for pulse outputs sent between the starting and stopping positions. The range is from -32768 to +32767 encoder counts. The time between count-based outputs must be at least as great as the servo period, whether these are array

value in RND mode or intervals in INC mode. The time will be the count spacing divided by the speed in count/second.

time-based number (optional)

Sets the number of time-based pulses that are generated after each encoder-count based pulse. The range is from 0-512 pulses.

time-based period (optional)

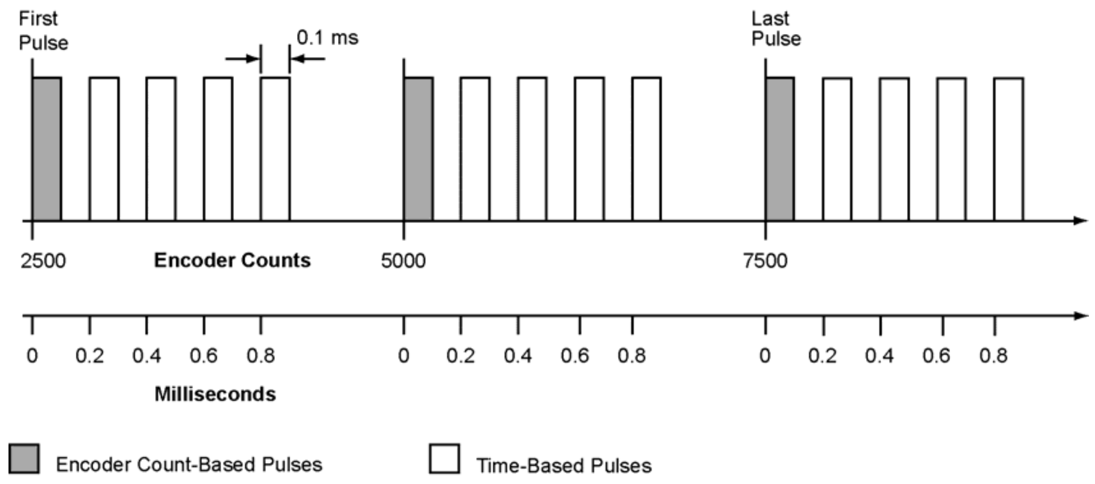
Sets the spacing of time-based pulses. For the ACR1505, the range is 0.1 – 2000000 µs. For the ACR9000/9030, the allowable spacing is 0.040 – 40000000 µs, with an actual resolution of 0.020 µs. The spacing must be at least 0.020 µs greater than the pulse width.

Example

The following example is setting up an incremental pulse sequence on Axis 0 output.

The **OOP** command starts an incremental **HSINT** sequence starting at 2500 counts (Start Pulse) and ending at 7500 counts (Stop Pulse), with a count-based output every 2500 counts between. After every count-based pulse, a time-based sequence of four (4) pulses is generated at 200 µs intervals. The output pulse width is set to 10 µs.

```
OOP 0 INC (10,2500,7500,2500,4,200)
```



OOP OFF Output On Position - Termination

		Product	Revision
Format	OOP *axis* OFF	ACR1505	1.18.15
Group	Feedback Control	ACR8020	n/a
Units	Microseconds	ACR9000	1.24
Data Type	Long Array	ACR9030	1.24
Default	Null	ACR9040	n/a
Prompt Level	SYS, PROGx		
See Also	IHPOS, INT, INTCAP, MSEEK, PLS	Aries CE	n/a
Related Topics	N/A		

Description

Used when the high-speed output is enabled and the user wants to terminate the function before completion.

Remarks

This command is valid for both incremental (**INC**) and random (**RND**) **OOP** modes.

When **OOP** is used, the output control flag (such as Bit 32) no longer controls the physical output. To return control of the physical output to the appropriate control flag, issue `OOPn OFF` (for Bit 32, this would be `OOP0 OFF`).

When **OOP** is controlling the physical output, the output control flag (such as Bit 32) does not indicate the status of the physical output.

Example

The following immediately terminates the high-speed output for Axis 0.

```
OOP 0 OFF
```

OOP RND Output On Position - Random Mode

		Product	Revision
Format	OOP *axis* RND(*width*, {*time-based number*, *time-based period*})		
Group	Feedback Control		
Units		ACR1505	1.18.15
Data Type	See above	ACR8020	n/a
Default	See above	ACR9000	1.24
Prompt Level	SYS, PROGx	ACR9030	1.24
See Also	IHPOS, INT, INTCAP, MSEEK, OOP BASE, PLS	ACR9040	n/a
Related Topics	N/A	Aries CE	n/a

Description

Uses an array of encoder count data points at which outputs are to be generated.

Remarks

The spacing of the points in the array depends on the product. For the ACR1505, up to four position-based outputs can be defined per servo period within the array data. For the ACR9000/9030, only one position-based output can be defined per servo period within the array data.

The data position in the array must follow in the strict ascending or strict descending order. The direction of the motion must also correspond with the order of the array data. If these conditions are not met, then erroneous pulses may occur at the output.

Optionally, in addition to the position-based output pulses, the ACR controllers can generate user-specified time-based pulses. These time-based pulses follow each position-based pulse. If the time-based components of the command are omitted, then the ACR controller does not generate time-based pulse outputs.

When **OOP** is used, the output control flag (such as Bit 32) no longer controls the physical output. To return control of the physical output to the appropriate control flag, issue `OOPn OFF` (for Bit 32, this would be `OOP0 OFF`).

When **OOP** is controlling the physical output, the output control flag (such as Bit 32) does not indicate the status of the physical output.

Arguments

axis	Specifies which axis output to use. For ACR1505, there are only two high-speed axes—Axis 0 and Axis 1. For the ACR9000/9030, each axis has an OOP which can be configured for either normal speed, 1.3 ms, or high speed, 0.20 µs (for OOP 0-2 and 4-6 only, not applicable to OOP 3 and 7).
mode	Determines which mode to use. For incremental mode, type `INC`. For random mode, type `RND`.

width Sets the generated pulse width in microseconds. The ACR1505, the range is 0.033 – 1000 µs. For the ACR9000/9030, the allowable pulse widths are 0.020 to 40000000 µs, with a resolution of 0.020 µs.

start Sets the position for the first pulse. The position is in encoder counts or steps.

stop Sets the position for the last pulse. The position is in encoder counts or steps.

interval
Sets the encoder-count interval for pulse outputs sent between the starting and stopping positions. The range is from -32768 to +32767 encoder counts. The time between count-based outputs must be at least as great as the servo period, whether these are array value in RND mode or intervals in INC mode. The time will be the count spacing divided by the speed in count/second.

time-based number (optional)
Sets the number of time-based pulses that are generated after each encoder-count based pulse. The range is from 0-512 pulses.

time-based period (optional)
Sets the spacing of time-based pulses. For the ACR1505, the range is 0.1 – 2000000 µs. For the ACR9000/9030, the allowable spacing is 0.040 – 40000000 µs, with an actual resolution of 0.020 µs. The spacing must be at least 0.020 µs greater than the pulse width.

Example

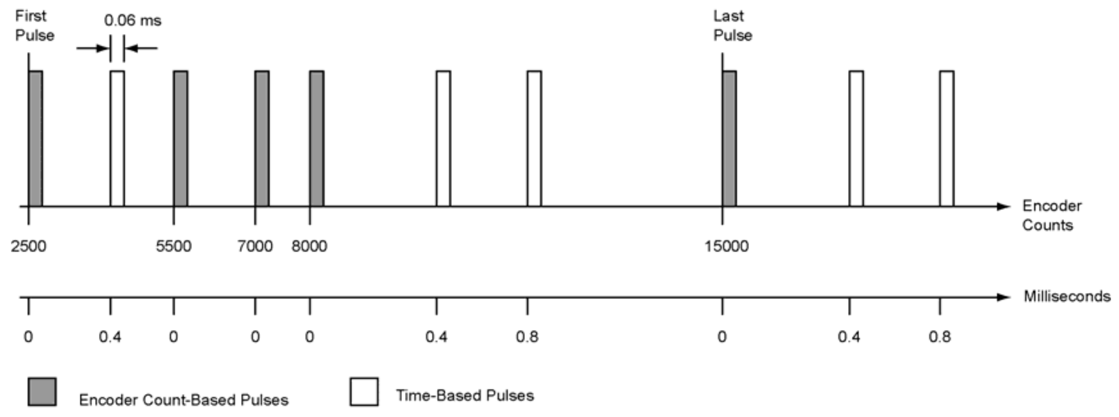
The following example is setting up a random pulse sequence on Axis 1 output.

The **OOP** command starts a random **HSINT** sequence based on the data stored in array LA0. After every count-based pulse, a time-based sequence of two (2) pulses is begun. They are generated at 400 µs intervals when the timing is appropriate. The output pulse width is set to 60 µs.

```
DIM LA1
DIM LA0(4) : REM  The BASE array has to be multiple of 4

LA0(0)=2500
LA0(1)=5500
LA0{2)=7000
LA0(3)=8000

OOP 1 BASE LA0
OOP 1 RND (60,2,400)
```


First Pulse
0.06 ms
Last Pulse
Encoder Counts
2500
5500
7000
8000
15000
Milliseconds
0
0.4
0
0
0
0.4
0.8
0
0.4
0.8
Encoder Count-Based Pulses
Time-Based Pulses

OPEN Open a Device

Format	OPEN "*device string*" AS #*device*
Group	Character I/O
Units	None
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	CLOSE, INPUT, OPEN DTALK, PRINT
Related Topics	N/A
Product Revision	Command and Firmware Release

This command opens a device. The valid range for *device* is zero through three. Each program has its own device zero which is used as its default device. Devices one through three are controller-wide system resources that can be opened and used from within any program or from any system or program prompt.

The "device string" describes the device that is to be opened. Serial device strings contain information required to set up communications.

When a device is opened, the operating system attached to that device enters an idle state, allowing incoming characters to be used by a program instead of being interpreted as commands. When the device is closed, the device enters its auto-detect mode as if starting from power-up.

Device Strings

You can use the following device strings. For streams, *n* represents the stream number which you want to open—stream 1 is dedicated to USB, streams 2-5 are for Ethernet.

"FIFO:"

"DPCB:"

"COM1:baudrate,parity,databits,stopbits"

"COM2:baudrate,parity,databits,stopbits"

“STREAMn:”

“EPLDn:”

NOTE: The device string cannot contain spaces.

NOTE: The **OPEN “***EPLDn***”** statement only works with controllers and drives that have the Ethernet Powerlink feature.

Following are the valid arguments for the device strings.

Baudrate: 300, 600, 1200, 2400, 9600, 19200, or 38400

Parity: N, E, or O

Databits: 5, 6, 7, or 8

Stopbits: 1 or 2

Example—RS232/485

The following example opens communication on COM1, prints "Hello world!" and then closes COM1.

```
OPEN "COM1:9600,N,8,1" AS #1
PRINT #1, "Hello world!"
CLOSE #1
```

Example—DPCB

The following example opens PCI bus communication, prints "Hello world!" and then closes.

```
OPEN "DPCB:" AS #1
PRINT #1, "Hello world!"
CLOSE #1
```

Example—FIFO

The following example opens ISA bus communication, prints "Hello world!" and then closes.

```
OPEN "FIFO:" AS #1
PRINT #1, "Hello world!"
CLOSE #1
```

Example—Ethernet

The following example opens communication on stream 2, prints "Hello world!" and then closes the stream.

```
OPEN "STREAM2:" AS #1
PRINT #1, "Hello world!"
CLOSE #1
```

Example—USB

The following example opens USB communication on stream 1, prints "Hello world!" and then closes.

```
OPEN "STREAM1:" AS #1
PRINT #1, "Hello world!"
CLOSE #1
```

Example—EPL Drive

The following example opens Ethernet Powerlink communication on stream 1, prints "Hello world!" and then closes.

```
OPEN "EPLD1:" AS #1
PRINT #1, "Hello world!"
CLOSE #1
```

OPEN DTALK Open a Drive Talk Device

Format	OPEN DTALK "device string" AS #*device*
Group	Character I/O
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	CLOSE, INPUT, OPEN, PRINT
Related Topics	N/A
Product Revision	Command and Firmware Release

NOTE: This command applies only to the ACR9000 controller.

This command opens a device. The valid range for *device* is zero through three. Each program has its own device zero which is used as its default device. Devices one through three are controller-wide system resources that can be opened and used from within any program or from any system or program prompt.

The "device string" describes the device that is to be opened. Serial device strings contain information required to set up communications.

When a device is opened, the operating system attached to that device enters an idle state, allowing incoming characters to be used by a program instead of being interpreted as commands. When the device is closed, the device will enter its auto-detect mode as if it were starting from power-up.

NOTE: The **OPEN DTALK** command only works with controllers and drives that have the Drive Talk feature.

Device Strings

You can use the following device strings.

"*COM1:baudrate,parity,databits,stopbits*"

"*COM2:baudrate,parity,databits,stopbits*"

NOTE: The device string cannot contain spaces.

Following are the valid arguments for the device strings.

- *Baudrate*: 300, 600, 1200, 2400, 9600, 19200, or 38400
- *Parity*: N, E, or O
- *Databits*: 5, 6, 7, or 8
- *Stopbits*: 1 or 2

Example

```
OPEN DTALK "COM2:9600,N,8,1" AS #1
DTALK X
CLOSE #1
```

OPEN EPLD Open an ETHERNET Powerlink Drive Connection

Format	OPEN EPLD “device string” AS #*device*
Group	Character I/O
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	DIAG_EPLD, EPLC
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Opens a Pseudo-Stream to an ETHERNET Powerlink drive (EPLD)

Remarks

A Pseudo-Stream provides a link between the ACR9030/9040’s TCP/IP client and user programs and external hosts. It is not a true stream and there is no way to enter commands on this stream. The name of the Pseudo-Stream is “EPLDn,” where “n” is the EPLD object number, with a possible range of 0-15. (See the *ACR9000 Hardware Installation Guide*, Appendix H, for more information on the Pseudo-Stream.)

“Device string” is the EPLD object number with the valid range from 0 through 15. When this command is used, “device string” is first tested to be sure it is less than the number of external nodes given by P37376. The ACR9030/9040’s TCP/IP client then makes a connection to the corresponding EPLD and links all input and output from that connection to the Pseudo-Stream.

Example

This example establishes a Pseudo-Stream connection to EPLD1 as #1, opens a Talk To channel, and then closes the connection.

```
OPEN “EPLD1:” AS #1
TALK TO #1
CLOSE #1
```

PARTNUMBER Partnumber

Format	PARTNUMBER *command* {“*string*”}
Group	Non volatile
Units	N/A
Data Type	String
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	N/A
Related Topics	N/A
Product Revision	Not available for Aries CE

Description

Queries the part number assigned by the factory.

Remarks

This command cannot be placed into a user program.

Example

Example

```
SYS>PARTNUMBER
9000P3U4B0
```

PASSWORD Password

Format	PASSWORD *command* {"*string*"}
Group	Operating System
Units	N/A
Data Type	String
Default	OFF
Prompt Level	SYS, PROGx, PLCx
See Also	CLEAR, ERASE, FLASH, LIST
Related Topics	N/A
Product Revision	Command and Firmware Release

The password feature is to lock certain commands so that once the password is on these commands can't be used.

PASSWORD OFF Password is Turned Off

Format	PASSWORD OFF {"*string*"}
Group	Operating System
Units	N/A
Data Type	STRING
Default	Off
Prompt Level	SYS, PROGx, PLCx
See Also	CLEAR, ERASE, FLASH, LIST
Related Topics	N/A
Product Revision	Command and Firmware Release

This command with password string will turn off the password. The password will remain off until the user turns it on again.

Example

The following example sets a password for program four, and then tries to list the program. When the password is given, you can access the program again.

```
P04>password on "abcdef"
P04>list
Wrong Password
P04>password off "abcdef"
```

PASSWORD ON Password is Turned On

Format	PASSWORD ON {"*string*"}
Group	Operating System
Units	N/A
Data Type	STRING
Default	Off
Prompt Level	SYS, PROGx, PLCx
See Also	CLEAR, ERASE, FLASH, LIST
Related Topics	N/A
Product Revision	Command and Firmware Release

This command allows you to set a password string at the system, program, or PLC level. The controller allows you to set a password only if the password protection is turned off. The password can be any ASCII string from 6 to 16 characters.

- When enabled at the system level, a user cannot list or upload programs or PLCs from the controller.
- When enabled at the program or PLC level, a user cannot list or upload to that particular program or PLC.

Once on, the password is in effect even if you cycle power. The **FLASH ERASE**, **CLEAR**, and **ERASE** commands cannot turn off the password. To turn off password protection, a user must correctly issue the **PASSWORD OFF** command.

NOTE: The password does not prevent users from modifying a program; it only prevents upload.

Example

The following example sets a password for program four, and then tries to list the program. When the password is given, you can access the program again.

```
P04>password on "abcdef"
P04>list
Wrong Password
P04>password off "abcdef"
```

PAUSE Activate Pause Mode

Format	PAUSE {PROG *number* | ALL}
Group	Program Control
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	AUT, BLK , HALT, RESUME, RUN, STEP, TROFF, TRON
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Pauses the currently selected program.

Argument

PROG *number* | ALL

Remarks

When you use **PAUSE**, the program's "pause control" bit is set. If there is no master attached, the "pause mode" bit is set as soon as the "pause control" is detected. Otherwise, the program will feedhold and then set "pause mode" when the master "in feedhold" is detected.

The **PAUSE** command pauses both programs and master motion, but does not pause **JOG**, **GEAR**, or **CAM**.

Master "cycle start requests" and **STEP** commands are ignored while in pause mode.

The **PAUSE PROG** command will pause the corresponding program and the **PAUSE ALL** command will pause all programs. These commands can be issued from anywhere in the system, including programs.

Example

The following pauses the current program:

```
PAUSE
RESUME   : REM Restart current program
```

Example Bits

Flag Parameter 4128 For Program 0	
Pause Control	1048
Pause Mode	1050

Flag Parameter 4112 For Program 0	
In Feedhold	519

PBOOT Auto-Run Program

Format	PBOOT
Group	Nonvolatile
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	BRESET, ELOAD, ERASE, ESAVE
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Allows user programs and PLCs to run automatically on power-up. When power is applied to the card (controller), the operating system checks the beginning of all programs and PLCs for the **PBOOT** command and sets their run request flag if the command is present.

Remarks

The power-up **PBOOT** check can also be initiated manually by issuing the **PBOOT** command from anywhere in the system. This allows the **PBOOT** sequencing to be tested without resetting or removing power to the card (controller).

NOTE: The **PBOOT** command must be the first line of code in a PLC or user program (after the command **PROGRAM**), or as the first line number. Otherwise, the controller cannot run the program at power-up.

Remark statements using **REM** are considered valid code and should not be placed before the **PBOOT** command. Remark statements using the apostrophe (') are stripped on download, and would not count as a line of code.

Examples

Example 1

The following program will run on power-up, flashing output 32 (on and off for a half second each):

```
PROGRAM
PBOOT
_LOOP1
BIT 32 = NOT BIT 32
DWL 0.500
GOTO LOOP1
ENDP
```

Example 2

The following will not execute on power-up because there is a REM before the PBOOT command.

```
PROGRAM
REM run the program at start_up
PBOOT
```

```
_LOOP1
BIT 32 = NOT BIT 32
DWL 0.100
GOTO LOOP1
ENDP
```

Example 3

This program will execute on start-up because a remark started with an apostrophe (') is not downloaded.

```
PROGRAM
‘ run the program at start-up
PBOOT
_LOOP1
BIT 32 = NOT BIT 32
DWL 0.100
GOTO LOOP1
ENDP
```

PERIOD Set Base System Timer Period

Format	PERIOD {*time*}
Group	Operating System
Units	Seconds
Data Type	FP32
Default	See below
Prompt Level	SYS, PROGx, PLCx
See Also	CPU, DIAG
Related Topics	N/A
Product Revision	Command and Firmware Release

This command set the base system timer period, the heartbeat of most operations involving the servo loops and motion profiling. Make sure to do a **CPU** command to check on the system load before lowering the period, dropping the period too low may cause sluggish foreground behavior, and loss of communication.

The recommended maximum value for the background percentage is 60% of the period, leaving at least 40% of the period for the processor to perform system tasks, such as program execution and serial/PC Bus communication. However, these recommended foreground and background percentages are generalized, and may be different based on individual system applications.

The valid range for base system timer period is 200 microseconds to 1 millisecond.

The default timer period is 500 microseconds for the ACR1200, ACR1505, ACR2000, ACR8000, ACR8010, ACR8020, and ACR9000 controllers.

The default timer period is 750 microseconds for the ACR1500 controller.

NOTE: Changing the period will affect motor tuning.

Example

The following example sets the base system period to 200 microseconds.

```
PERIOD 0.0002
```

PGAIN Set Proportional Gain

Format	PGAIN {*axis* {*value*}} {*axis* {*value*}} ...
Group	Servo Control
Units	volts / pulses of error
Data Type	FP32
Default	0.00244141
Prompt Level	PROGx
See Also	AXIS, DGAIN, FFACC, FFVEL, IGAIN
Related Topics	Axis Parameters (0-7) (8-15)
Product Revision	Command and Firmware Release

This command modifies the value used in the PID algorithm to control proportional gain. Issuing a **PGAIN** command to an axis without an argument will display the current setting for that axis. The default gain is 0.0024414 (10 volts at 4096 pulses) for all axes.

Example

The following example sets the X axis integral gain to 0.001 volts/pulse:

```
PGAIN X0.001
```

PLC Switch to a PLC Program Prompt

Format	PLC *number*
Group	Operating System
Units	None
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	PROG, SYS
Related Topics	N/A
Product Revision	Command and Firmware Release

This command switches the communication channel to the designated PLC program prompt. The *number* argument determines which PLC program you want to view. You cannot issue the **PLC** command without an argument; otherwise, it displays an error.

The communications channel must be at a PLC level in order to run and edit PLC programs. The PLC command cannot be issued from within a program.

Example

The following example switches from the system level to PLC program three. If you issue the **PLC** command without the number argument, the controller displays an error.

```
SYS>PLC 3
PLC3>
PLC3>PLC
Syntax Error
PLC3>
```

PLS Programmable Limit Switch

Format	PLS *index command* {*data*}
Group	Global Objects
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	CLEAR, DIM, SRC
Related Topics	N/A
Product Revision	Command and Firmware Release

This command is used with a second command to control the eight Programmable Limit Switch (PLS) objects that execute in the background. A PLS uses a source parameter to generate a table index. This table index is used to lookup an array entry that is then transferred into a destination parameter.

By pointing the source to an encoder position and the destination to the digital outputs, the PLS can sequence through a set of outputs based on a shaft position, similar to a mechanical cam operating a bank of switches.

The following is a list of valid PLS command combinations:

PLS BASE: Attach array to PLS.

PLS DST: Set PLS destination pointer.

PLS FLZ: Set PLS index offset.

PLS MASK: Set PLS output bit mask.

PLS OFF: Disable PLS update.

PLS ON: Enable PLS update.

PLS RATIO: Set PLS scaling ratio.

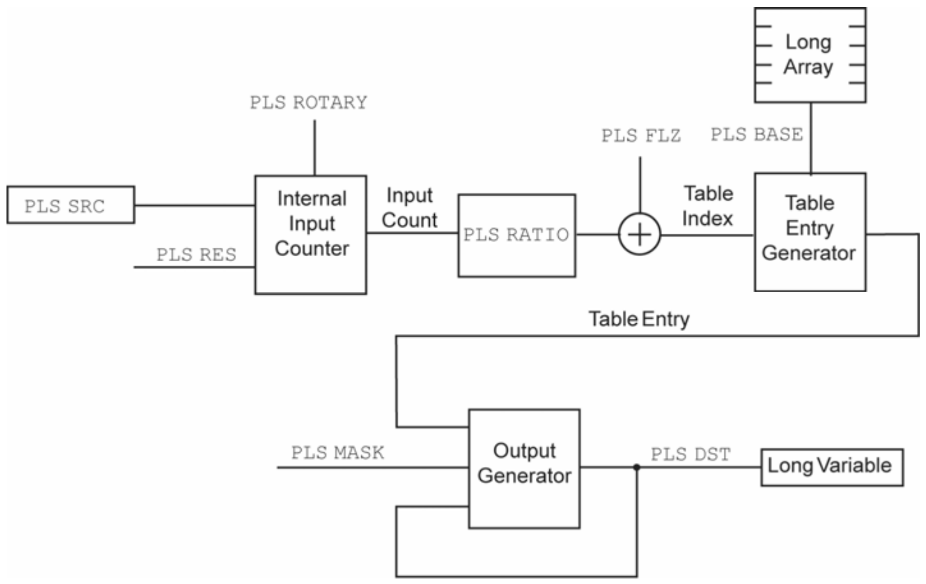
PLS RES: Reset or preload counter.

PLS ROTARY: Set PLS rotary length.

PLS SRC: Set PLS input source.

Since there are eight PLS objects, the *index* argument must be in the range of 0 to 7.

The following block diagram provides an example of PLS operation:



As the source changes, an internal input count is generated. If a rotary length is set with the **PLS ROTARY** command, the input count "wraps around" based on the rotary length. The input count can be reset or preloaded using the **PLS RES** command.

The internal input count is multiplied by the **PLS RATIO** and added to the **PLS FLZ** to generate a table index. The table index is used to fetch an entry from the long integer array pointed to with the **PLS BASE** command. If the table index is outside of the array boundaries, a zero is used instead of an array entry.

The table entry is merged with the parameter pointed to by the destination pointer. By default, the destination pointer points to P4097, the opto-isolated digital outputs, but this can be changed by using the **PLS DST** command. The PLS can be set to modify any or all of the 32 bits in the destination parameter by using the **PLS MASK** command.

The operating system updates the programmable limit switch every servo period.

Example 1

The following example sets PLS 0 to look at ENC 0 and use the long integer array LA0 for its table of values. The mask is set to 65535 (0x0000FFFF) so that only the lower 16 bits of P4097 (OUT32 - OUT43) will be modified. The bits will sequence through a binary pattern representing the encoder position from of 0 to 1999 pulses.

These examples assume that PROG0 has enough user memory allocated inside it to accommodate a 2000 element long integer array. This memory allocation can be done using the **DIM** command from the SYS level.

```
PROG0
DIM LA(1)
DIM LA0(2000)
DIM LV(5)
LV0=0
_LOOP1
LA0(LV0) = LV0
LV0=LV0+1
```

```
IF (LV0 < 2000) GOTO LOOP1
PLS0 SRC ENC0
PLS0 BASE LA0
PLS0 MASK 65535
PLS0 ON
RUN
```

Example 2

Issuing just the **PLS** command will display the current setting of a PLS and can be used even if the PLS is currently active. The example below shows the list:

```
P00>PLS0
PLS0 BASE LA2
PLS0 FLZ 100
PLS0 MASK 121
PLS0 RATIO 2.34
PLS0 ROTARY 202
PLS0 SRC CLOCK
PLS0 ON
```

PLS BASE Set PLS Array Pointer

Format	PLS *index* BASE LA *number*
Group	Global Objects
Units	none
Data Type	Number= Integer
Default	-1= no array
Prompt Level	PROGx
See Also	ADC, AXIS, DAC, DIM, ENC, PLS
Related Topics	N/A
Product Revision	Command and Firmware Release

This command sets the PLS array pointer. The *number* argument indicates which long integer array is to be used. The array must first be allocated using the **DIM** command. Since there is no default for the PLS array pointer, one must be defined before the **PLS ON** command is issued.

Example

The following example attaches PLS 4 to the LA2 array:

```
DIM LA(3)
PLS4 BASE LA2
```

PLS DST Set PLS Destination Pointer

Format 1:	PLS *index* DST LV *number*
Format 2:	PLS *index* DST P *number*
Group	Global Objects
Units	None
Data Type	N/A
Default	P4907
Prompt Level	PROGx
See Also	ADC, AXIS, DAC, DIM, ENC, PLS
Related Topics	N/A
Product Revision	Command and Firmware Release

This command sets the PLS destination pointer. The two command formats allow any long integer parameter to be used as the PLS destination. The default PLS destination pointer is the address of P4097, the opto-isolated digital output parameter.

Example 1

The following example sets the destination of PLS 3 to P4109 (XIO-2 outputs):

```
PLS3 DST P4109
```

Example 2

The following example set the destination of PLS0 to local long variable 1:

```
DIM LV(2)
PLS0 DST LV1
```

PLS FLZ Set PLS Index Offset

Format	PLS *index* FLZ {*offset*}
Group	Global Objects
Units	array entries
Data Type	Long
Default	0
Prompt Level	PROGx
See Also	ADC, AXIS, DAC, ENC, MASK, PLS
Related Topics	N/A
Product Revision	Command and Firmware Release

This command sets the index offset. Issuing a **PLS FLZ** command with no argument will display the current setting. The default index offset is 0 array entries.

Example

The following example sets the offsets PLS 3 by 10 array entries:

```
PLS3 FLZ 10
```

PLS MASK Set PLS Output Bit Mask

Format	PLS *index* MASK {*mask*}
Group	Global Objects
Units	none
Data Type	Long
Default	-1 (0xFFFFFFFF)
Prompt Level	PROGx
See Also	ADC, AXIS, DAC, ENC, PLS
Related Topics	N/A
Product Revision	Command and Firmware Release

This command sets the PLS output bit mask. Issuing a **PLS MASK** command with no argument will display the current setting. The default mask setting is -1 (0xFFFFFFFF) allowing all bits to be transferred.

When the table entry is transferred to the location defined by the **PLS DST** command, the bit mask is used to determine which bits will be transferred. The bit transfer is done according to the following logic formula:

dest = (*dest* AND NOT *mask*) OR (*entry* AND *mask*)

Example

The following example sets the mask for PLS 5 to 255, enabling the lower eight bits of the destination:

```
PLS5 MASK 255
```

PLS OFF Disable PLS Update

Format	PLS *index* OFF
Group	Global Objects
Units	none
Data Type	N/A
Default	OFF
Prompt Level	PROGx
See Also	ADC, AXIS, DAC, ENC, PLS
Related Topics	N/A
Product Revision	Command and Firmware Release

This command disables **PLS** update. The **PLS** output will remain in the state that it was when the **PLS** was turned off.

Example

The following example disables the update of PLS 3:

```
PLS3 OFF
```

PLS ON Enable PLS Update

Format	PLS *index* ON
Group	Global Objects
Units	none
Data Type	N/A
Default	OFF
Prompt Level	PROGx
See Also	ADC, AXIS, DAC, ENC, PLS
Related Topics	N/A
Product Revision	Command and Firmware Release

This command enables **PLS** update. After this command is sent, the internal PLS counter increments with the source.

Example

The following example enables the update of PLS 4:

```
PLS4 ON
```

PLS RATIO Set PLS Scaling Ratio

Format	PLS *index* RATIO {*ratio*}
Group	Global Objects
Units	array entries / input count
Data Type	FP32
Default	1
Prompt Level	PROGx
See Also	ADC, AXIS, DAC, ENC, PLS
Related Topics	N/A
Product Revision	Command and Firmware Release

This command sets the scaling ratio. Issuing a **PLS RATIO** command with no argument will display the current setting. The default index offset is 1.0 array entries/count.

Because there are no fractional, array, entry indexes, the Table Index (see diagram in PLS command) value is always rounded down.

Example

The following example sets the scaling ratio of PLS 2 to 0.25 entries / count:

```
PLS2 RATIO 0.25
```

PLS RES   Reset or Preload Internal Counter

Format	PLS *index* RES {*offset*}
Group	Global Objects
Units	input counts
Data Type	Long
Default	0
Prompt Level	PROGx
See Also	ADC, AXIS, DAC, ENC, PLS
Related Topics	N/A
Product Revision	Command and Firmware Release

This command resets or preloads the **PLS** internal input counter. If the *offset* argument is left out, the counter is set to zero. The internal counter is only used if the **PLS** is set to rotary operation by the **PLS ROTARY** command. In a linear PLS (PLS rotary =0), the internal input count is always equal to the source.

Example

The following example resets the internal counter on PLS 7 to 1000 input source counts:

```
PLS7 RES 1000
```

PLS ROTARY Set PLS Rotary Length

Format	PLS *index* ROTARY {*length*}
Group	Global Objects
Units	input counts
Data Type	Long
Default	0
Prompt Level	PROGx
See Also	ADC, AXIS, DAC, ENC, PLS
Related Topics	N/A
Product Revision	Command and Firmware Release

This command sets the PLS rotary length. Issuing a **PLS ROTARY** command with no argument will display the current setting. The default rotary length is 0 counts.

If the rotary length is zero, the PLS is linear and the output will be zeroed if a table index is generated that lies outside the boundaries of the PLS array. The internal input count is always equal to the current value of the source parameter when a PLS is linear.

If the rotary length is non-zero, the source parameter is used to generate an input count that "wraps-around" by the given length (modulus) before it is used to generate a table index. Note that this only affects the internal PLS input count and that it is still possible to generate table indexes outside of the array boundaries. While in rotary mode, the count can be modified with the **PLS RES** command.

Example

The following example sets PLS 6 rotary length to 2000 counts:

```
PLS6 ROTARY 2000
```

PLS SRC Set PLS Input Source

Format	PLS *index* SRC *sourcedef*
Group	Global Objects
Units	None
Data Type	N/A
Default	None
Prompt Level	PROGx
See Also	SRC
Related Topics	N/A
Product Revision	Command and Firmware Release

This command specifies the source for the input of a PLS. See the SRC command for the definition of the *sourcedef* argument. The default source is set to none.

NOTE: The **PLS SRC** is never scaled by the PPU of the axis to which the encoder is attached.

Example

The following example sets the source of PLS 5 to encoder 3:

```
PLS5 SRC ENC3
```

PM ACC Position Maintenance Acceleration

Format	PM ACC {*axis* {*units*}}
Group	Axis Limits
Units	units/second2 scalable by PPU
Data Type	FP32
Default	100
Prompt Level	SYS, PROGx
See Also	PM ACC, PM LIST, PM OFF, PM ON, PM REN, PM SCALE, PM VEL
Related Topics	N/A
Product Revision	Command and Firmware Release

This command sets the acceleration for the corrective move.

Example

The following example sets up and then enables position maintenance.

```
EXC X 100
PM DB X 10
PM VEL X 100
PM ACC X 500
PM SCALE X 2
PM ON X
```

PM DB Position Maintenance Deadband

Format	PM DB {*axis* {*encoder_counts*}}
Group	Axis Limits
Units	Encoder Counts; Minimum= 2
Data Type	LONG
Default	10
Prompt Level	SYS, PROGx
See Also	PM ACC, PM LIST, PM OFF, PM ON, PM REN, PM SCALE, PM VEL
Related Topics	N/A
Product Revision	Command and Firmware Release

This command sets the deadband zone where, when the axis enters the region, position maintenance stops. The Position Maintenance Active flag clears when an axis enters its assigned deadband range.

All **PM DB** values are entered in stepper counts.

Example

The following example sets up and then enables position maintenance.

```
EXC X 100
PM DB X 10
PM VEL X 100
PM ACC X 500
PM SCALE X 2
PM ON X
```

PM LIST Position Maintenance Settings

Format	PM LIST {*axis*}
Group	Axis Limits
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx
See Also	PM ACC, PM DB, PM OFF, PM ON, PM REN, PM SCALE, PM VEL
Related Topics	N/A
Product Revision	Command and Firmware Release

This is a general-purpose command, and displays the current setting of position maintenance feature.

Example

The following example sets up and then enables position maintenance.

```
PROG0> PM LIST X
EXC X 100
PM DB X 10
PM VEL X 100
PM ACC X 500
PM SCALE X 2
PM ON X

PROG0>
```

PM OFF Position Maintenance Disable

Format	PM OFF {*axis*}
Group	Axis Limits
Units	N/A
Data Type	N/A
Default	Off
Prompt Level	SYS, PROGx
See Also	PM ACC, PM DB, PM LIST, PM ON, PM REN, PM SCALE, PM VEL
Related Topics	N/A
Product Revision	Command and Firmware Release

This command will disable the position maintenance for an axis.

The following table shows the results for various command sequences. It assumes that the drive and position maintenance are enabled (**DRIVE ON X, PM ON X**).

Command Sequence			
Sequence 1	**Sequence 2**	**Sequence 3**	**Sequence 4**
DRIVE OFF X	PM OFF X	PM OFF X	PM OFF X
—	—	DRIVE OFF X	DRIVE OFF X
Manually move Axis.	Manually move Axis.	Manually move Axis.	Manually move Axis.
DRIVE ON X	PM ON X	DRIVE ON X	PM ON X
—	—	PM ON X	DRIVE ON X
Result			
Holds Encoder Actual Position	Holds Commanded Position	Holds Commanded Position	Holds Encoder Actual Position

Example

The following example disables position maintenance on axis X.

```
PM OFF X
```

PM ON Position Maintenance Enable

Format	PM ON {*axis*}
Group	Axis Limits
Units	N/A
Data Type	N/A
Default	OFF
Prompt Level	SYS, PROGx
See Also	PM ACC, PM DB, PM LIST, PM OFF, PM REN, PM SCALE, PM VEL
Related Topics	N/A
Product Revision	Command and Firmware Release

This command enables position maintenance for an axis. If actual position and commanded position differ and that difference is larger than the position maintenance deadband (**PM DB**), then the controller generates a corrective move.

NOTE: With stall detect enabled, stalls in the motor are recognized using the bounds set in the **EXC** command. Any value outside the boundaries results in a stall.

NOTE: Do not confuse position maintenance for true servoing. Position maintenance is only invoked at the end of a move—it continually monitors the position and corrects when necessary. Servoing takes place throughout the entire move—servos continually make adjustments on the fly.

The following table shows the results for various command sequences. It assumes that the drive and position maintenance are enabled (**DRIVE ON X, PM ON X**).

Command Sequence			
Sequence 1	**Sequence 2**	**Sequence 3**	**Sequence 4**
DRIVE OFF X	PM OFF X	PM OFF X	PM OFF X
—	—	DRIVE OFF X	DRIVE OFF X
Manually move Axis.	Manually move Axis.	Manually move Axis.	Manually move Axis.
DRIVE ON X	PM ON X	DRIVE ON X	PM ON X
—	—	PM ON X	DRIVE ON X
Result			
Holds Encoder Actual Position	Holds Commanded Position	Holds Commanded Position	Holds Encoder Actual Position

Example

The following example sets up and then enables position maintenance.

```
EXC X 100
PM DB X 10
PM VEL X 100
PM ACC X 500
PM SCALE X 2
PM ON X
```

PM REN Position Maintenance Match Position With Encoder

Format	PM REN {*axis* {*position_in_units*}}
Group	Axis Limits
Units	Units scalable by PPU
Data Type	FP32
Default	NA
Prompt Level	SYS, PROGx
See Also	MULT, PM ACC, PM DB, PM LIST, PM OFF, PM ON, PM SCALE, PM VEL, REN
Related Topics	N/A
Product Revision	Command and Firmware Release

This command causes the controller to find the position maintenance scale and the encoder **MULT** sign for the respective axis.

NOTE: The **PM REN** command needs several motor revolutions to make the correct determination.

Example

The following example moves axis X one thousand units, providing enough distance for the **PM REN** command to determine the scaling and **MULT** signs.

```
PM REN X 1000
```

PM SCALE Position Maintenance Scale

Format	PM SCALE {*axis*} {*ratio*}
Group	Axis Limits
Units	Ratio (Stepper Pulses / Encoder Counts)
Data Type	FP32
Default	1.0
Prompt Level	SYS, PROGx
See Also	PM ACC, PM DB, PM LIST, PM OFF, PM ON, PM REN, PM VEL
Related Topics	N/A
Product Revision	Command and Firmware Release

This command sets a value equal to the ratio between stepper pulses and encoder counts per revolution.

Example

The following example sets up and then enables position maintenance.

```
EXC X 100
PM DB X 10
PM VEL X 100
PM ACC X 500
PM SCALE X 2
PM ON X
```

PM VEL Position Maintenance Velocity

Format	PM VEL {*axis* {*units*}}
Group	Axis Limits
Units	units/second scalable by PPU
Data Type	FP32
Default	100
Prompt Level	SYS, PROGx
See Also	PM ACC, PM DB, PM LIST, PM OFF, PM ON, PM REN, PM SCALE
Related Topics	N/A
Product Revision	Command and Firmware Release

This command sets the maximum velocity for a corrective move.

Example

The following example sets up and then enables position maintenance.

```
EXC X 100
PM DB X 10
PM VEL X 100
PM ACC X 500
PM SCALE X 2
PM ON X
```

PPU Set Axis Pulse/Unit Ratio

Format	PPU {*axis* {*ratio*}} {*axis* {*ratio*}} ...
Group	Feedback Control
Units	pulses / unit
Data Type	FP32
Default	1
Prompt Level	PROGx
See Also	AXIS, JOG, MASTER, RES, REN
Related Topics	Axis Parameters (0-7) (8-15)
Product Revision	Command and Firmware Release

Description

This sets the pulses per programming unit (PPU) for each axis. It allows programming in inches, millimeters, degrees, revolutions, or other units. Issuing a **PPU** command to an axis without an argument displays the current setting for that axis.

Remarks

Do not set the pulses per unit as a round number. Instead, express it as a ratio (for example, 4000/360 for rotational motion). A ratio provides several advantages: It makes your code more readable, allowing you to quickly determine how the PPU is derived; it increases accuracy because the DSP is able to take advantage of floating point numbers; and it avoids potential floating point rounding errors.

NOTE: Changing the **PPU** affects the axis velocity profile as compared to its master; therefore, master parameters such as **VEL** and **ACC** may need to be adjusted before changing the **PPU** of an axis. **EXC** and **IPB** may also have to be adjusted if **PPU** is changed.

NOTE: Do not use a negative **PPU** value.

PPU values are set by the ACR-View configuration wizard in the Scaling screen. The **ESAVE** command saves the PPU values, and they are stored in Axis Parameters:

		Mask	0x01	0x02	0x04	0x08	0x10	0x20	0x40	0x80
Index	**Reserved Parameters Code=0x30**		**Axis Number**							
			0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0x57	PPU	FP32	12375	12631	12887	13143	13399	13655	13911	14167

Index	**Reserved Parameters Code=0x30**		**Axis Number**							
			0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0x57	PPU	FP32	14423	14679	14935	15191	15447	15703	15959	16215

Examples

Example 1

The following example assumes a 1000 line encoder attached to a motor. The **MULT** command brings the value to 4000. **PPU X4000** then sets the programming units to revolutions (4000 pulses/rev) for the rest of the program. The X axis will move 200 revolutions at 20 revs/second, using 10 revs/second² ramps.

```
MULT X4
PPU X4000
ACC 10 DEC 10 STP 10
VEL 20
X/200
```

Example 2

The following commands are valid at any prompt or program:

```
SYS>PPU AXIS0
4000
SYS>PPU AXIS1
5000
```

The following commands are valid only at the program prompt or in a program where the axis is attached:

```
P00>PPU X
4000
P00>PPU Y
5000
```

Example 3

If the desired programming units are encoder pulses, set PPU equal to 1.

```
AXIS0 PPU 1
```

Example 4

Assume a motor has a 2000-line encoder (8000 pulses per revolution post-quadrature). The motor is connected to a 5 mm/revolution ballscrew. The desired programming units are millimeters.

8000 pulses per rev / 5 mm per rev = 1600 pulses per mm

To set the PPU, you can use:

```
PPU X1600
```

Or you may use expressions and generic axis designations:

```
AXIS0 PPU (8000/5)

PROGRAM
PPU X1600
RES X   : REM set position counters to zero
ACC 500 STP500 DEC 500 VEL 100
X10   : REM move 10mm
INH -516   : REM wait for move to complete
PRINT P12290   : REM display actual position in encoder counter
PRINT (P12290/P12735)   : REM display actual position in user units
REM by using PPU parameter
ENDP

LRUN
15998
9.99875
```

Example 5

Assume a motor has a 2000-line encoder (8000 pulses per revolution post-quadrature), and a belt drive actuator with a 3.386-inch diameter drive pulley with an 8:1 gearbox. The desired programming units are inches:

1 motor rev = (3.386 inches x 3.1416) / 8 = 1.3297 inches / rev

To set the PPU, you can use:

```
PPU X 6016.47
```

Or you may use expressions and generic axis designations:

```
AXIS0 PPU (8*8000/(3.386*3.1416))
```

PRINT Send Data to a Device

Format	PRINT {#*device*,} {USING "*format_string*" ;} {*expressionlist*}
Group	Character I/O
Units	None
Data Type	N/A
Default	Device= 0
Prompt Level	SYS, PROGx, PLCx
See Also	INPUT, OPEN, CLOSE
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

This command prints a series of expressions (parameter, bit, and string values) to a device. If no device number is given, device zero is used. If the device is closed, or was never opened, the **PRINT** command is ignored. The LISTEN and LRUN commands will temporarily open device zero, allowing normal **PRINT** output to be monitored.

Remarks

When **PRINT** is used in conjunction with **USING**, the "format string" defines the format of numeric output. In the format string, a number sign (#) represents each digit. A plus sign (+) at the beginning of the string forces a sign for positive numbers. An optional decimal point (.) will print the number in decimal format.

Numbers are rounded as necessary. The output width will be equal to the length of the format string, right justified and padded with spaces. An exception to this rule occurs when a number is too big to fit in the defined format string. The following are examples of legitimate format strings:

"###.####" Three leading, four trailing digits

"+###.##" Three leading, two trailing, forced sign

"####" Four digits total, no decimal point

The *expressionlist* argument is a list of expressions separated by either commas (,)or semicolons (;). The comma inserts a tab character between the expressions, and a semicolon indicates there is no space between the expressions. A print statement that does not end with either a comma or a semicolon produces a carriage return/linefeed combination.

NOTE: As a shortcut in the terminal, you can use a question mark (?) in place of the **PRINT** command.

Examples

Example 1

This will print text to the terminal when using the **LRUN** or **LISTEN** command.

```
PROG0
PROGRAM
P1=100.1234    : REM Global Variable assignment
PRINT “Hello World”
PRINT””    : REM blank line
PRINT “Program Running”
PRINT “Line Number=”;P7168    : REM P7168 is Program0 current line number
PRINT “Variable Value=”;P1
PRINT “Commas”,”tab”,”to”,”next”,”tab”,”stop”
PRINT “Variable Value=”,P1
ENDP

P00>LRUN
Hello World

Program Running
Line Number=60
Variable Value=100.1234
Commas  tab     to      next    tab     stop
Variable Value=         100.1234
P00>
```

Example 2

```
DIM DV(1)
DIM $V(1,10)
DV0 = 5678
$V0 = "ABC"
OPEN "COM2:19200,N,8,1" AS #1
PRINT #1,
PRINT #1, 1234; DV0, $V0;
PRINT #1, "DEF"
PRINT #1, "Outputs = "; P4097
PRINT #1, USING "###.####"; 1/SQRT(2)
PRINT #1, USING "+##.####"; COS(45)
CLOSE #1
```

Example 3

The following example demonstrates the *expressionlist* argument.

```
P00> PRINT “WATER”
WATER
P00>PRINT “WATER”,
WATER   P00>
P00>PRINT “WATER”;
WATERP00>
```

PROG Switch to a Program Prompt

Format	PROG {*number*}
Group	Operating System
Units	None
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	ATTACH, PLC, SYS
Related Topics	N/A
Product Revision	Command and Firmware Release

This command switches the communication channel to the designated program prompt. Issuing a **PROG** command without an argument will either display the current program number or display an error (if not at a program level.)

The prompt keeps track of your current program or system level as follows:

SYS>PROG3

P03>PROG1

P01>SYS

SYS>_

The communications channel must be at a program level in order to run and edit programs. The **PROG** command cannot be issued from within a program.

Example

The following example demonstrates how you must include a program number, otherwise the controller returns an error. It switches from the system prompt to program four, then provides a report back.

```
SYS>PROG
NOT AT PROGRAM LEVEL
SYS>PROG4
P04>PROG
4
```

PROGRAM Beginning Of Program Definition

Format	PROGRAM
Group	Program Flow
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	ENDP, GOSUB, GOTO, RUN
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Marks the start of a program that does not use line numbers.

Remarks

The **ENDP** command marks the end of a program that does not use line numbers.

If the **ENDP** command is not included, then immediate mode commands are not executed. Instead, the commands are stored in the program space as well.

NOTE: When using the **LIST** command to view a lineless program, you can view the automatic line numbers by setting bit 5651. (**SET 5651**)

Example

```
SYS
HALT ALL
NEW ALL
CLEAR
DIM PROG0(1000)
DIM DEF 10
#DEFINE LED BIT96
#DEFINE myflag BIT 32
#DEFINE TRUE 1

PROG0
#DEFINE Counter LV2
#DEFINE loop LV4

PROGRAM
DIM LV10
Counter=0
SET myflag
WHILE (myflag)
     Counter= Counter+1
     FOR loop=100 TO 500 STEP 200
          PRINT loop ;",";
     NEXT
     GOSUB SetLED
     IF (Counter>5)
          CLR myflag
          PRINT " Done "
          ELSE
          PRINT " Busy "
     ENDIF
WEND
```

```
PRINT “END OF PROGRAM,”
END

_SetLED
SET LED
PRINT “ LED on “; LED,
RETURN

ENDP
```

PROM Dump Burner Image

Format	PROM *index* {#*device*}
Group	Nonvolatile
Units	None
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	ELOAD, ESAVE, ERASE, PBOOT
Related Topics	N/A
Product Revision	Command and Firmware Release

This command is not valid for the ACR1500 controller.

This command dumps a burner image of the onboard executive plus the user programs and PLC programs. By burning these images into EPROM and replacing the EPROMs on the boards, the card will load programs from the EPROM on power up instead of relying on battery backup memory. User variables always reside in battery backup memory and are not affected by the program transfer.

The *index* argument selects the image for the EPROMs as follows:

Index	ACR8000
0	U10
1	U11

The optional *device* argument allows the image to be dumped to an open channel like the **PRINT** command does. By using the **OPEN** and **CLOSE** commands, an EPROM burner could be connected directly to an unused serial port.

The image is dumped in "Intel MCS-86 Hex Object" format for direct download into an EPROM burner. The image can also be captured for later download. The burner must be capable of burning 4 megabit (256K x 16) devices, typically from the 27C4096 family. The device must be 175 nanoseconds or faster for normal operation and 95 nanoseconds or faster for the optional 1 wait state mode for the ACR8000. The device must be 85 nanoseconds or faster for the ACR1200, ACR2000, ACR8010 (Recommended device: ATMEL P/N AT27C4096-85JC).

Example

The following example dumps the EPROM image for U10 on the ACR8000 daughterboard.

```
OPEN "COM1:38400,n,8,1" AS #1
PROM 0 #1
CLOSE #1
```

RATCH Software Ratchet

Format	RATCH *index command* {*data*}
Group	Global Objects
Units	None
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	CAM, GEAR, JOG, PLS, SRC
Related Topics	N/A
Product Revision	Command and Firmware Release

This command is used along with a second command to setup software ratchets. The ratchet *index* is a number from 0 to 7. Software ratchets are sources that can ignore, negate, or buffer both positive and negative pulses.

When a ratchet is set up for buffering, pulses in the buffering direction are added to an internal count instead of causing the ratchet output to change. Pulses in the normal direction are first used to unbuffer previously buffered pulses. When there are no more pulses to unbuffer, the ratchet tracks normally.

The following is a list of valid ratchet command combinations:

RATCH MODE: Set ratchet mode.

RATCH SRC: Define ratchet source.

RATCH MODE Set Ratchet Mode

Format	RATCH *index* MODE {*mode*}
Group	Global Objects
Units	none
Data Type	N/A
Default	0
Prompt Level	PROGx
See Also	ADC, AXIS, DAC, ENC
Related Topics	N/A
Product Revision	Command and Firmware Release

This command sets the conversion mode for a ratchet. Issuing a mode command to a ratchet without an argument will display the current mode for that ratchet. The default ratchet mode is zero.

The following is a table of ratchet modes and their affect on incoming source pulses:

Mode	Positive Pulses	Negative Pulses
0	Normal	Normal
1	Normal	Ignore
2	Normal	Negate
3	Normal	Buffer
4	Ignore	Normal
5	Ignore	Ignore
6	Ignore	Negate
7	Ignore	Buffer
8	Negate	Normal
9	Negate	Ignore
10	Negate	Negate
11	Negate	Buffer
12	Buffer	Normal
13	Buffer	Ignore
14	Buffer	Negate
15	Buffer	Buffer

Normal: Incoming pulses directly change ratchet output.

Ignore: Incoming pulses are ignored and do not change the ratchet output.

Negate: Incoming pulses are negated then directly change ratchet output.

Buffer: Incoming pulses are buffered and added to the internal count, but do not change the ratchet output. Pulses in the normal direction are first used to unbuffer previous pulses.

Example

The following example sets ratchet 7 to buffer negative pulses:

```
RATCH7 MODE 3
```

RATCH SRC Define Ratchet Source

Format	RATCH *index* SRC *sourcedef*
Group	Global Objects
Units	none
Data Type	N/A
Default	None
Prompt Level	PROGx
See Also	SRC
Related Topics	N/A
Product Revision	Command and Firmware Release

This command sets the input source for a ratchet. The default ratchet source is “none”. See the SRC command for the definition of the *sourcedef* argument.

Example

The following example sets the source of ratchet 2 to encoder 7:

```
RATCH2 SRC ENC7
```

REBOOT Reboot Controller Card

Format	REBOOT
Group	Operating System
Units	None
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	RUN, HALT
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Reboots the controller in the same manner as pressing the reset button.

Remarks

This also shuts down communications, turns off outputs, kills programs, and anything else that a hardware reset or power cycle does.

Example

The following reboots the card:

```
REBOOT
```

REM Program Comment

Format	REM *comment*
Group	Program Flow
Units	None
Data Type	N/A
Default	N/a
Prompt Level	PROGx
See Also	END, ENDP, PROGRAM, RUN
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Allows you to add explanatory remarks to a program, or prevent the execution of a statement.

Argument

comment	Any text you want to include

Remarks

Used at the beginning of a line, a space is required between the **REM** command and the *comment* text. Program execution continues with the first line of executable statement following the **REM** statement.

To add a comment to a line of code, a space, colon (:), and another space must precede the **REM** command.

Comments cannot be put on the same line as program labels.

Comments take up memory on the controller, and can affect the controller's processing time.

You may use an apostrophe (') in place of the **REM** command at the beginning of a line, and these comments will be stripped away during download. Apostrophes may not be used in place of the **REM** command to add a comment to a line of code.

As a troubleshooting technique, insert the **REM** command at the beginning of a line of code to remove that line from the program run.

Example

The following uses **REM** to prevent code execution for specific lines:

```
PROGRAM
REM This program illustrates the REM command
REM This code is not executed due to the REM command
PRINT "This line is executed."
PRINT "This line is also executed."
ENDP
```

Output:

```
This line is executed.
This line is also executed.
```

The following shows the **REM** command being used in troubleshooting:

```
PROGRAM
REM This program illustrates use of the REM command in troubleshooting
PRINT “This line is executed.”
REM PRINT “This line is also executed.”
ENDP
```

Output:

```
This line is executed.
```

The following shows how to add a **REM** comment to a program line. The program line still executes.

```
ACC 10000 : REM This a comment following an executable statement
```

The following uses an apostrophe to prevent code execution for specific lines:

```
’ This is a comment that will be removed during download
```

The follow shows improper use of the **REM** command:

```
_Label1 : REM Do not put REM commands on the same line as a program label
```

The following shows improper use of the apostrophe (’):

```
PRINT “BAD CODE” : ’ You may not use apostrophes to put remarks after code
```

REN Match Position with Encoder

Format	REN {*axis*} {*axis*} ...		
Group	Feedback Control		
Units	None		
Data Type	N/A		
Default	N/A		
Prompt Level	PROGx		
Report Back	None	**To view, type**	N/A
See Also	AXIS, JOG, REN, RES		
Related Topics	Axis Flags (0-7) (8-15), Master Flags (0-7) (8-15)		
Product Revision	Command and Firmware Release		

Description

Removes following error by adjusting the commanded position to equal the actual position.

Arguments

axis	Assigns which axes are being reset to actual position.

Remarks

The **REN** command copies the actual position from the encoder to the secondary setpoint in the servo loop. The servo loop calculates backwards to determine the primary setpoint and current position. The action, in effect, removes position error. The **REN** command does not affect the values of the Jog, Cam, and Gear offsets

Suppose the motor power is off and a motor is manually turned. You can send a **REN** before motor power is applied; the controller learns the new position and zeroes out the digital/analog command signal.

NOTE: For the ACR9000 controller, the **DRIVE ON** command automatically performs a **REN** operation.

Example

The following example resets the commanded position for axes X, Y, and Z with their respective actual feedback positions.

```
REN X Y Z
```

RES Reset or Preload Encoders

Format	RES {*axis* {*preload*}} {*axis* {*preload*}} ...
Group	Feedback Control
Units	Units based on PPU
Data Type	FP32
Default	0
Prompt Level	PROGx
Report Back	None **To view, type** N/A
See Also	AXIS, CAM RES, CAM SRC RES, GEAR RES, JOG RES, REN, PPU
Related Topics	Encoder Flags
Product Revision	Command and Firmware Release

Description

Sets the commanded and actual position to zero.

Arguments

axis

Optional. Assigns an axis.

preload

Optional. Assigns a position you want loaded into the commanded and encoder position registers.

Remarks

The **RES** command zeros out the primary setpoint, which includes the coordinated motion, Jog, Gear, and Cam motion profilers. You can also preload positions into the commanded and actual position registers.

You can also use **RES** to zero out the primary setpoint for SSI encoders.

NOTE: The **RES** command does not clear the actual position register for analog feedback devices.

Example

The following example zeros the X axis commanded and actual position registers and pre-loads Y axis to 1000 units.

```
RES X Y1000
```

RESUME Release Pause Mode

Format	RESUME {PROG *number* | ALL}
Group	Program Control
Units	NONE
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	AUT, BLK, HALT, PAUSE, RUN, STEP, TROFF, TRON
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Resumes the currently selected program by clearing the program's "pause control" bit.

Arguments

PROG number	Program number 0 through 15
ALL	All programs currently paused

Remarks

When the current program's "pause control" bit is cleared, a master "cycle start request" is issued to the attached master, the program's "pause mode" bit is cleared, and the program resumes.

The following is a list of valid command combinations:

RESUME PROG	Resumes the corresponding program.
RESUME ALL	Resumes all programs.

These commands can be issued from anywhere in the system, including programs.

Example

The following resumes execution of program 1:

```
P01> RESUME
```

The following resumes execution of program 2:

```
RESUME PROG2
```

The following resumes all paused programs:

```
RESUME ALL
```

Example Bits

Program Flags

Flag Parameter	4128	4129	...	4143
	Program 0	**Program 1**	...	**Program 15**
Pause Control	1048	1080	...	1528
Pause Mode	1050	1082	...	1530

Master Flags

Flag Parameter	4112	4113	...	4119
	Master 0	**Master 1**	...	**Master7**
Cycle Start Request	521	553	...	745

Flag Parameter	4328	4329	...	4335
	Master 8	**Master 9**	...	**Master 15**
Cycle Start Request	7433	7465	...	7657

RETURN Return from a Subroutine

Format	RETURN
Group	Program Flow
Units	None
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	GOSUB, GOTO
Related Topics	N/A
Product Revision	Command and Firmware Release

This command causes an unconditional return from a subroutine. Program execution will continue at the command following the last **GOSUB** command that was executed. An error will occur if a **RETURN** is executed without a prior **GOSUB** command.

Example

```
REM main program
PRINT "Entering the main program"
GOTO LOOP1
PRINT "Leaving the main program"
END
REM first subroutine
_LOOP1
PRINT ".Entering first subroutine"
GOTO LOOP2
PRINT ".Leaving first subroutine"
RETURN
REM second subroutine
_LOOP2
PRINT "..Entering second subroutine"
PRINT "..Leaving second subroutine"
RETURN
LRUN
```

Example Output

```
Entering the main program
.Entering first subroutine
..Entering second subroutine
..Leaving second subroutine
.Leaving first subroutine
Leaving the main program
```

ROTARY Set Rotary Axis Length

Format	ROTARY {*axis* {*length*}} {*axis* {*length*}} ...
Group	Feedback Control
Units	Units based on PPU
Data Type	FP32
Default	0
Prompt Level	PROGx
See Also	AXIS, NORM, RES, REN, PPU
Related Topics	N/A
Product Revision	Command and Firmware Release

This command sets the rotary axis length used for the shortest distance calculations. Issuing a **ROTARY** command without an argument will display the current setting. The default rotary length is 0.0 for all axes, disabling shortest distance moves.

If the rotary length of an axis is non-zero, a **MOD** function is performed on absolute moves and the result is run through a shortest distance calculation. The resulting move will never be longer than half the rotary axis length. Incremental moves are not affected by the rotary axis length.

This procedure actually converts absolute moves into incremental moves that are up to plus or minus half the rotary length. Current positions are normally generated that lie outside of the rotary length boundaries. The **NORM** command can be used to return the current position to within the bounds of the rotary length.

To increase the accuracy of the rotary motion, reset the **PPU** value to degrees using a ratio (for example **PPU X (4000/360)**). This has an added benefit—it allows you to quickly change the PPU in your program if you replace a motor in the application. It is important to note that you must reset the encoder (**RES**) and error band (**EXC**) to values appropriate to the new PPU value.

Example

The following example sets the rotary length of the A axis to 360 units:

```
ROTARY A360
A120    : REM move to 120 units results in positive motion
A0    : REM go back to 0 position
A275
REM move to 275 units results in negative motion as this is the
REM shortest distance.
```

ROTATE Rotate A Programmed Path

Format	ROTATE *rotate_angle primary* {*rotate_center*} *secondary* {*rotate_center*}
Group	Transformation
Units	Rotate angle= degrees; rotate center=units based on PPU
Data Type	FP32
Default	N/A
Prompt Level	PROGx
See Also	FLZ, MASTER, OFFSET, SCALE
Related Topics	N/A
Product Revision	Command and Firmware Release

This command will cause the programmed path to be rotated about the given center point. If the rotational center for an axis is not specified, the rotational center for that axis is equal to its current location.

The "primary" and "secondary" axes define the plane of rotation. Positive rotation is from the primary axis towards the secondary axis.

Example

```
ROTATE 30 X1 Y2
```

ROV Set Rapid Feedrate Override

Format	ROV {*rate*}
Group	Velocity Profile
Units	Multiplier of velocity
Data Type	FP32
Default	1
Prompt Level	PROGx
Report Back	Yes **To view, type ROV**
See Also	FOV, MASTER, VEL
Related Topics	Master Parameters (0-7) (8-15), Secondary Master Flags (0-7) (8-15)
Product Revision	Command and Firmware Release

Description

Scales the velocity for the current master profile.

Arguments

rate

Optional. Sets the percentage change to the **VEL** rate for the master profile. The rate is a multiplier, and only affects the coordinated motion profiler.

Remarks

The **ROV** command has two bits that control when **ROV** is active—"Rapid Pending" and "Rapid Active." The new velocity, as changed by **ROV**, remains in effect until you send another **ROV** command. The master uses the current **ACC** or **DEC** rates to ramp to the new velocity.

The **ROV** command affects only the coordinated motion profiler, and does not affect the Jog, Gear, or Cam profilers.

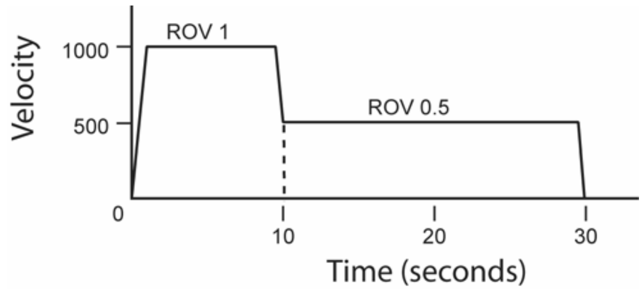
- Set the "Rapid Pending" bit to enable **ROV** at the start of the next move. The start of the next move sets the "Rapid Active" bit. To stop **ROV** from affecting the next move, clear the "Rapid Pending" bit.
- Set the "Rapid Active" bit to enable **ROV** immediately for the current move. To disable **ROV**, clear the "Rapid Active" bit.

NOTE: The **FOV** and **ROV** commands behave similarly. While the effect of **ROV** is the same as **FOV**, it has two bits controlling when it becomes active. Furthermore, when **ROV** is enabled **FOV** is disabled.

NOTE: When the "Rapid Active" bit is set, **ROV** is enabled and **FOV** is disabled.

Example

The following example moves at velocity for ten seconds, then reduces velocity to 50 percent for the remainder of the move:



```
ROV 1    : REM ensures is set to default
DEC 1000 ACC 1000 STP 1000 VEL 1000
X/20000    : REM incremental move
DWL 10     : REM dwell for ten seconds
SET 2051    : REM sets Rapid Active bit for axis 0
ROV 0.5    : REM scales velocity to 50 percent
```

RUN Run a Stored Program

Format 1:	RUN {*line*}
Format 2:	RUN {PROG *number* {*line*} | PLC *number* | ALL}
Group	Program Control
Units	None
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	HALT, LRUN, LISTEN, PLC, PROG
Related Topics	Program Parameters
Product Revision	Command and Firmware Release

Description

Runs the current program and returns to the command prompt.

Argument

Format 1

line	Specifies program line number where program execution begins.

Format 2

PROG *number*	Specifies program number.
line	Optional line number where program execution begins.
PLC *number*	Specifies PLC number.
ALL	Specifies all programs and all PLCs.

Remarks

The **RUN** command in Format 1 cannot be issued from inside a program.

Issuing a **RUN** command with the optional *line* argument will start program execution at the given line number.

The optional **RUN** formats (Format 2) can be issued from anywhere, including programs. The **RUN PROG** and **RUN PLC** commands will run the corresponding user or PLC program. The **RUN ALL** command will run all user and PLC programs.

Normally when a program is run, the communication channel returns to the command prompt, allowing more commands to be entered. While at the command prompt, output from programs (including error reporting) is shut down to prevent mixing of command input and program output.

During initial program testing, it is suggested that the **LRUN** command be used instead of the **RUN** command, allowing monitoring of the program output until an escape character (ASCII 27) is received or the program ends. Otherwise, the program may terminate but the reported error will not be seen by the user.

Example

The following runs program 1:

```
RUN PROG1
```

The following runs program 10 starting at line 100:

```
RUN PROG10 100
```

The following runs PLC 0:

```
RUN PLC0
```

SAMP Data Sampling Control

Format	SAMP {*channel*} *command* {*data*}
Group	Global Objects
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	AXIS, DAC, ENC, PERIOD, PLS
Related Topics	Miscellaneous Outputs, Miscellaneous Parameters
Product Revision	Command and Firmware Release

The sample command is combined with other commands to prepare the system for data sampling. Data sampling allows the monitoring of system parameters at the servo interrupt period. Optionally, data may be also sampled at fixed frequency (i.e. every 25 milliseconds) or on the rising or falling edge of a bit flag.

A total of eight channels (0-7) that can be simultaneously filled from different parameter sources. For example, one channel can monitor actual position while another is monitoring output voltage for a given axis. The resulting information can then be transferred to an offline system for graphical plotting or tuning analysis.

The following is a list of valid sample command combinations:

SAMP BASE: Set sample base.

SAMP CLEAR: Clear sample channels.

SAMP SRC: Set sample source.

SAMP TRG: Set sample trigger.

Code Execution Outline

```
if (sample trigger armed)
    if (trigger condition met)
        if ((level trigger) or (edge trigger and not trigger latched))
            set sample in progress
        set trigger latched flag
    else clear trigger latched flag
if (sample in progress)
    if (sample clock > 0)
        update sample clock
    if (sample clock = 0)
        sample clock = sample period
        sample active channels
        increment sample index
        if (channels full)
            clear sample armed flag
            clear sample active flag
            sample index = 0
        if (edge trigger)
            clear sample active flag
```

Example

The following example takes a 500 samples of axis 0 current position and output signal at the default servo interrupt rate of 2 kHz (250 milliseconds total sample time):

```
PROG0
ATTACH MASTER0
ATTACH SLAVE0 AXIS0 "X"
DIM LA1
DIM LA0(500)
DIM SA1
DIM SA0(500)
SAMP CLEAR    : REM reset sample defaults
SAMP0 SRC P12290    : REM axis 0 current position
SAMP0 BASE LA0    : REM store data in LA0
SAMP1 SRC P12319    : REM axis 0 output signal
SAMP1 BASE SA0    : REM store data in SA0
SAMP TRG 516    : REM master 0 in motion flag
SET 104    : REM arm the sample trigger
X1000    : REM move axis / start sample
INH -104    : REM wait for sample done
```

SAMP BASE Set Sample Base

Format 1:	SAMP *channel* BASE LA *index*
Format 2:	SAMP *channel* BASE SA *index*
Group	Global Objects
Units	None
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	AXIS, DAC, ENC, PLS, SAMP
Related Topics	N/A
Product Revision	Command and Firmware Release

This command selects a storage array base for the given sample channel. Sample channels are numbered 0 through 7. The 'base' parameter can either a 32-bit long integer array (LA) or a 32-bit floating point array (SA).

When sampling, the channel will transfer information into this array from the source set by the **SAMP SRC** command. The source and base should both be of the same type since no data conversion is done during the transfer.

Example

The following example ties sample channel 0 to the long integer array LA0 and then ties channel 1 to the 32-bit floating point array SA0:

```
SAMP0 BASE LA0
SAMP1 BASE SA0
```

SAMP CLEAR Clear Sample Channels

Format	SAMP CLEAR
Group	Global Objects
Units	None
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	AXIS, CLEAR, DAC, ENC, PLS, SAMP
Related Topics	N/A
Product Revision	Command and Firmware Release

This command clears out all of the system parameters and flags that are related to data sampling. It also clears out any the internal pointers that may have been set with the **SAMP SRC** and **SAMP BASE** commands.

Example

```
SAMP CLEAR
```

SAMP SRC Set Sample Source

Format	SAMP *channel* SRC *parameter*
Group	Global Objects
Units	None
Data Type	N/A
Default	None
Prompt Level	SYS, PROGx, PLCx
See Also	AXIS, CLEAR, DAC, ENC, PLS, SAMP, SRC
Related Topics	N/A
Product Revision	Command and Firmware Release

This command selects a source for the given sample channel. Sample channels are numbered 0 through 7. The source 'parameter' can be either a system parameter or any user defined parameter.

When sampling, the channel will transfer information from the source into the array set by the **SAMP BASE** command. The source and base should both be of the same type since no data conversion is done during the transfer.

Example

The following example sets **SAMP0** source to **AXIS0** actual position (P12290) and **SAMP1** source to **AXIS0** output signal (P12319):

```
SAMP0 SRC P12290
SAMP1 SRC P12319
```

SAMP TRG Set Sample Trigger

Format 1:	SAMP TRG + *index*
Format 2:	SAMP TRG - *index*
Group	Global Objects
Units	None
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	AXIS, CLEAR, DAC, ENC, PLS, SAMP
Related Topics	N/A
Product Revision	Command and Firmware Release

This command sets the trigger condition to be monitored when the sample trigger armed flag is set. A positive index will cause a trigger to occur on an active state or a rising edge, depending on the setting of the sample mode flag. A negative index will cause a trigger on an inactive state or a falling edge.

Example

The following example will start sampling when MASTER0 starts moving (rising edge of Bit516):

```
SAMP TRG516
```

SCALE Scale A Programmed Path

Format	SCALE *ratio* {*axis* {*center*}} {*axis* {*center*}} ...
Group	Transformation
Units	Center=units based on PPU; ratio=none
Data Type	FP32
Default	N/A
Prompt Level	PROGx
See Also	FLZ, OFFSET, ROTATE
Related Topics	N/A
Product Revision	Command and Firmware Release

This command will cause the programmed path to shrink or expand about the given center point. If the "center" for an axis is not specified, the scaling center for that axis is equal to its current location.

Example

```
SCALE 0.5 X Y
SCALE 2 Z1.5
```

SET Set a Bit Flag

Format	SET *index*
Group	Logic Function
Units	None
Data Type	Boolean
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	BIT, CLEAR, IF THEN, INH
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Sets (turns on or activates) the specified bit flag.

Argument

index

Remarks

This flag can either be a physical output or an internal bit.

Example

The following example pulses output 32 for 2 seconds.

```
SET 32
DWL 2
CLR 32
```

SINE Sinusoidal Move

Format	SINE {*axis*(*target*, phase, sweep, amplitude)} ...		
Group	Interpolation		
Units	Sweep and phase=degrees; target and amplitude=units based on PPU		
Data Type	FP32		
Default	N/A		
Prompt Level	PROGx		
Report Back	None	**To view, type**	N/A
See Also	AXIS, MOV, TRJ, PPU		
Related Topics	Axis Flags (0-7) (8-15),		
Product Revision	Command and Firmware Release		

Description

Generates a sinusoidal profile.

Arguments

axis

Optional. Assigns an axis.

target

Required. Position at the end of the move (in units).

phase

Required. Sinusoidal phase shift (in degrees).

sweep

Required. Total number of degrees in the sinusoid.

amplitude

Required. Amplitude of the sinusoid (in units).

Remarks

When the move begins, the coordinated motion profiler generates an internal “current angle”, which starts at zero degrees and increases until reaching the *sweep* value. The formula used to determine the current position is as follows:

Current position = center point + amplitude * sin (phase + current angle)

NOTE: Using **SINE** with two axes lets you generate circles and ellipses.

Generating an arc

The current position of the axes is the starting position for sine motion. Using the starting position and the *target* and *center* arguments, the coordinated motion profiler calculates the sine motion using the following formulas:

xtheta1

xphase = theta1 + 90

xamplitude = radius

xtarget = end position for X axis

ysweep = theta2 - theta1

yphase = theta1

yamplitude = radius

ytarget = end position for Y axis

Where:

theta1 = start angle of arc

theta2 = end angle of arc

radius = radius of the arc

xcenter = xtarget - radius * cos(theta2)

xstart = xcenter + radius * cos(theta1)

ycenter = ytarget - radius * sin(theta2)

ystart = ycenter + radius * sin(theta1)

Caution — Be sure to move the axes to the correct starting point before the move begins. If the axes are not at the correct starting point, the axes can fault or jump to the starting point.

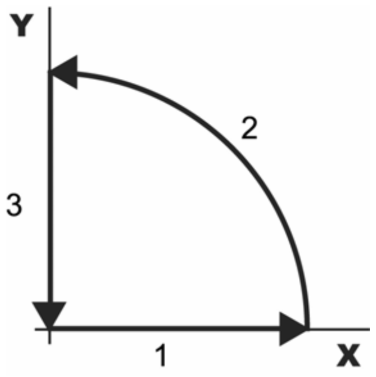
If the axes are not at the correct starting point—calculated from the SINE arguments—the coordinated motion profiler adjusts the phase or amplitude as necessary.

Example: Circular Interpolation

The following example generates a pie-shaped path in the first quadrant.

```
X0 Y0
X10000 Y0 : REM Move 1
SINE X(0,90,90,10000) SINE Y(10000,0,90,10000) : REM Move 2
```

X0 Y0 : REM Move 3



Example: Spiral Interpolation

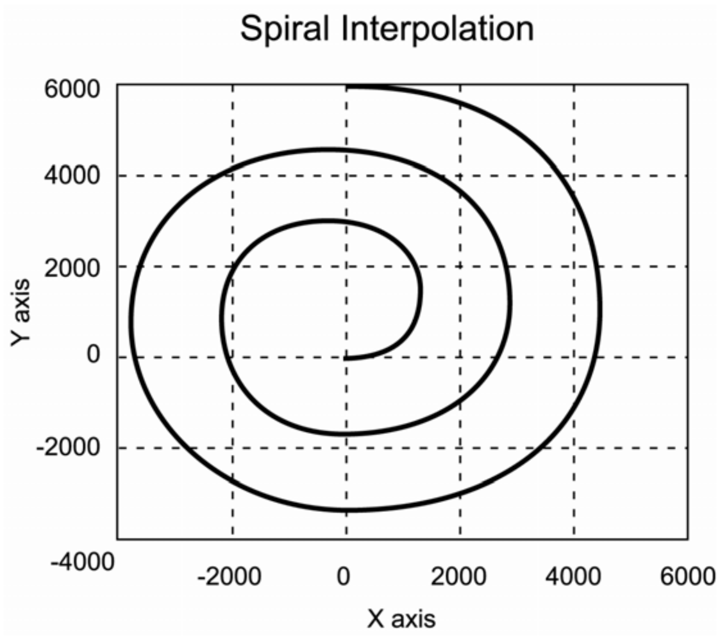
The sinusoidal interpolation can be extended to draw spiral shapes by specifying the start and end amplitude of the sine wave, which will become the start and end radius of the spiral. The secondary master flag "Spiral Mode" should be set to activate this spiral interpolation.

SINE {*axis* **(***target***,** *phase***,** *sweep***,** *start_amplitude***,** *end_amplitude***)}**

Example

This Spiral interpolation example generates a spiral with a start radius of 1000, end radius of 5000, sweep of 900 degrees and a target of (0,6000)

```
SET 2072 : REM master 0 secondary flag "spiral mode"
SINE X(0 ,0,900,1000,5000) SINE Y(6000,270,900,1000,5000)
```



SLDEC Software Limit Deceleration

Format	SLDEC {*axis* {*units*}}
Group	Axis Limits
Units	Units
Data Type	FP32
Default	0
Prompt Level	PROGx
See Also	SLIM, SLM
Product Revision	Command and Firmware Release

This command sets the rate of deceleration used after hitting a programmed software end-of-travel limit.

NOTE: If limits are enabled and the motor/load encounters a software limit, motion stops at the rate set by the **SLDEC** command. To clear the switch, motion must occur in the opposite direction.

Example

The following example set the deceleration for axis X to 50,000.

```
SLDEC X50000
```

SLIM Software Limit Enable

Format	SLIM *axis* {*value*}
Group	Axis Limits
Units	See Below
Data Type	N/A
Default	0
Prompt Level	PROGx
See Also	SLDEC, SLM
Related Topics	N/A
Product Revision	Command and Firmware Release

This command enables or disables the inputs defined as software end-of travel limits. With limits disabled, motion is not restricted.

When you enable a specific limit (positive or negative) and the limit wiring for the enabled limit is a physical open circuit, motion is restricted—the defaults for the Limit Level Invert (bits 22 and 23, Parameters 4600-4615) bits are set.

NOTE: If limits are enabled and the motor/load encounters a software limit, motion stops at the rate set by the **SLDEC** command. To clear the switch, motion must occur in the opposite direction.

SLIM Hardware Limits

Value	Description
0	Disables positive limit and negative limit (default)
1	Enables positive limit and disables negative limit
2	Disables positive limit and enables negative limit
3	Enables positive limit and negative limit

Example

The following example enables both positive and negative hardware limits for axis X.

```
SLIM X3
```

SLM Software Limit Positive/Negative Travel Range

Format	SLM {*axis*(*units*)} {*axis*(*high*, low)}
Group	Axis Limits
Units	Units scaleable by PPU
Data Type	FP32
Default	0
Prompt Level	PROGx
See Also	SLM
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

This command sets the positive and negative travel range (distance in absolute units) for software limits. When the commanded position of a given axis moves beyond a software limit, the axis, master, and its attached axes decelerate at the rate set by the **SLDEC** command.

Remarks

When the software end-of-travel limit is exceeded, the Master's Kill All Moves Request flag (for example, Bit 522) will be set which will prevent buffered interpolated moves (**MOV**, **CIRCW**, **CIRCCW**, **SINE**, **TARC**/**SPLINE**/**NURB**) from occurring.

Examples

Example 1

Sets the range of motion for axis X and Y to 50,000 counts.

```
PPU X4000 Y4000 : REM for example clarification only.
REM (ACR-View's Config Wizard sets this value)
SLM X12.5 Y12.5
SLIM X3 Y3 : REM enable software end-of-travel checking
```

Example 2

Sets the positive limit for axes X and Y to 50,000 counts, and the negative limit for axes X and Y to -40,000 counts.

```
PPU X4000 Y4000
SLM X(12.5,-10) Y(12.5, -10)
SLIM X3 Y3
```

SPLINE Cubic Spline Interpolation

Format	SPLINE *command*
Group	Interpolation
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	MASTER, NURB, TMOV
Related Topics	Tertiary Master Flags (0-7) (8-15), Master Parameters (0-7) (8-15)
Product Revision	Command and Firmware Release

NOTE: This command does not apply to the ACR8000 controller.

The Cubic Spline Interpolation fits a smooth curve, exactly passing through the data points specified by the user. The data points can be non-evenly spaced. This is based on the clamped Cubic Spline algorithms, thus allowing the user to specify the initial and final velocity in the algorithm

The following is a list of valid **SPLINE** command combinations:

SPLINE MODE: Enable **SPLINE** Interpolation Type.

SPLINE END: End **SPLINE** Interpolation.

The following is a typical single **SPLINE** command format for a 2-D curve with X and Y axes.

```
K 3 X5 Y/2 W2.3 VEL 5
```

Data	Description
K 3	Knot Value of 3
X5	Absolute target point for x-axis
Y / 2	Incremental target point for y-axis
VEL 5	velocity from previous knot to this knot

If the knot vectors are not included in the command, then the delta between knots is equal to the value set by **TMOV** command, where the first knot in Spline block is always equal to zero.

The **VEL** command is also optional, and ,if omitted, the previously used value is put into the next segments. The simplified Spline command would than look as follows:

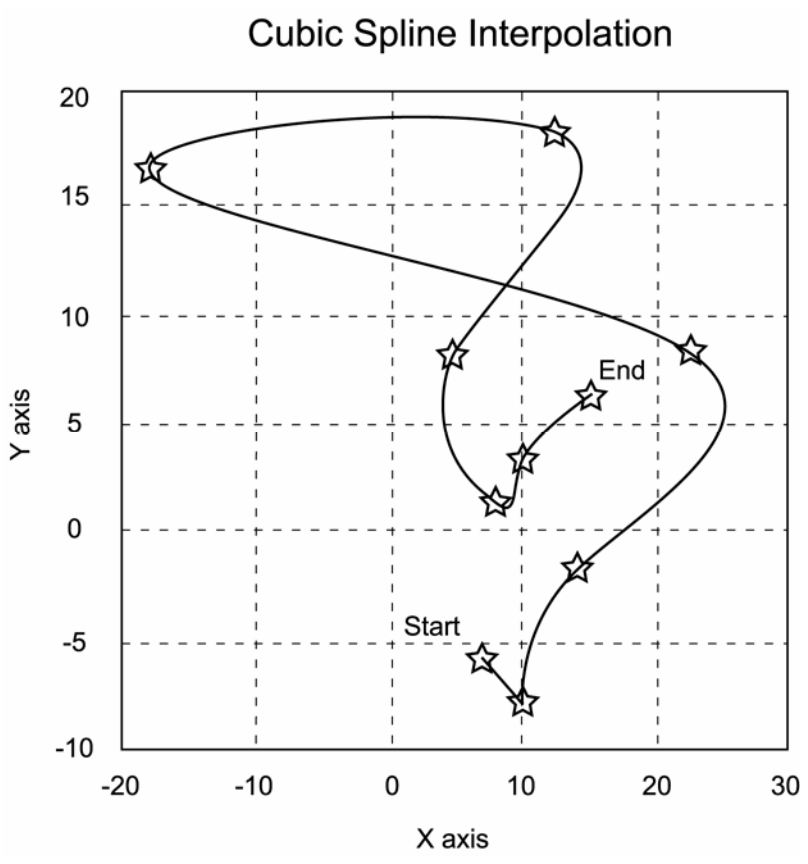
```
X5 Y/2
```

Data	Description
X5	Absolute target point for x-axis
Y / 2	Incremental target point for y-axis

NOTE: The Cubic Spline algorithm uses six data point to calculate the motion trajectory. Using the **INH** and **DWL** commands in the Spline block will make the motion stop four points before the place where these commands were issued.

NOTE: To ensure good results the data points should be smoothly spaced, which means that the data can be non-uniform but does not have abrupt changes.

NOTE: The axes must be at the location of the first point before turning on spline motion; otherwise, a continuity error occurs.



Example

The following example uses the target spline points as shown in the illustration above. The resulting Spline curve follows smooth and exactly through the spline points.

```
RES X Y
MOV X7 Y-6
INH -516
SPLINE MODE 0 : REM Time Based Mode Is Selected
K 0 X7 Y-6
REM If the knots are not included, then TMOV value is used as a delta REM
between two points. Note that first knot should be always zero.
K 3 X10 Y-8
K 6 X14 Y-2
K 8 X22 Y8
K 10 X-18 Y16 : REM Spline passes this point at exactly 10 seconds
K 12 X12 Y18
K 14 X4 Y8
K 16 X8 Y1
K 18 X10 Y3
K20 X15 Y6 : REM Spline passes this point at exactly 20 seconds
K20 X15 Y6 : REM Must be replicated four more times to anchor point
K20 X15 Y6
K20 X15 Y6
K20 X15 Y6
K -1 : REM Negative knot indicates that Spline block has ended
```

SPLINE END Ends Spline Interpolation Mode

Format	SPLINE END
Group	Interpolation
Units	None
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	MASTER, NURB
Related Topics	Tertiary Master Flags (0-7) (8-15)
Product Revision	Command and Firmware Release

This command is used to terminate the **SPLINE** interpolation mode initiated by the **SPLINE MODE** command.

The **SPLINE** ending is automatically done, if the **SPLINE** motion has normally come to end/stop by a negative knot. However, if stopped abnormally, like issuing incomplete or wrong data to the **SPLINE** profiler, then this command must be used to terminate the **SPLINE** mode.

Example

```
SPLINE END
```

SPLINE MODE Enable Spline Mode

Format	SPLINE MODE {*value*} ...
Group	Interpolation
Units	None
Data Type	N/A
Default	0
Prompt Level	PROGx
See Also	MASTER, NURB
Related Topics	Tertiary Master Flags (0-7) (8-15)
Product Revision	Command and Firmware Release

This command sets the card into Spline Interpolation mode. Each Spline block should start with this command. The subsequent move commands are treated as Spline target points and Spline interpolation is used to calculate the curve path. This modes remains active until a negative knot is received, Kill Move Flag is set, or the user issues a **SPLINE END** command.

SPLINE Interpolation Modes

To give control over machine speed following modes are available. The user decides which one to use, depending upon the application.

Mode	SPLINE Type	TOV	FOV	Description
0	Time Interpolation	Yes	NA	Automatic speed and acceleration control depending upon how the data points are in time. The master parameter Spline Time Factor may be used to scale the time axis, thereby changing the speed. Very smooth and good for high speed. Good for making time based moves.
1	Velocity Interpolation	Yes	NS	Linearly maps time to knots, where time between points is calculated by (Linear-Arc-length / **VEL**).
2	Velocity-Accel Interpolation	Yes	NS	Linearly maps time to knots, where time between points is the maximum of (Linear-Arc-length / **VEL**). And (2× Linear-Arc-length / **ACC**)$^{0.5}$
3	Velocity-**TMOV** Interpolation	Yes	NS	Linearly maps time to knots, where time between points is the maximum of (Linear-Arc-length / **VEL**). And **TMOV** time.$^{0.5}$
4	NR Interpolation	Yes	Yes	Newton Raphson approximation for vector velocity control.
NA:	Not Applicable			
NS:	**FOV** value is applied to the next segment			

Example

The following example selects Time Interpolation Mode.

```
SPLINE MODE 0
```

SRC Set External Timebase

Format	SRC *sourcedef*
Group	Velocity Profile
Units	none
Data Type	N/A
Default	Clock
Prompt Level	PROGx
See Also	CAM, ENC, EPLD, GEAR, LIMIT, MASTER, PERIOD, PLS, RATCH, RES, REN, SAMP
Related Topics	N/A
Product Revision	Command and Firmware Release

This command specifies the timebase for coordinated motion. The sourcedef can be defined in any of the following formats:

- ***NONE***: Disconnect device from source.
- ***CLOCK****:* Connect to servo clock (1 pulse per period).
- ***ENC*** *encoder number*: Connect to encoder register.
- ***EPLD*** *EPLD number*: Connect to EPLD register.
- ***LIMIT*** *limiter number*: Connect to frequency limiter output.
- ***RATCH*** *ratchet number*: Connect to ratchet output.
- ***P*** *parameter number:* Connect to user or system parameter.
- ***RES*** *{preload}*: Reset or preload internal source count.
- ***REN****:* Match internal source count to external input.

During each servo interrupt, the change in source pulses is multiplied by the servo period and the resulting delta time is fed into the velocity profile mechanism. By default, the velocity profile is sourced off the clock, feeding a single time unit per interrupt. Redirecting the source allows an external timebase to be used for coordinated motion.

NOTE: When using *ENC encoder* or *encoder* arguments, you can only use encoders zero through seven. To access encoders eight and nine, use the *parameter* argument.

NOTE: For homing operations, always use the clock as the source.

Example

The following example sets source of the current master to ratchet number 5:

```
SRC RATCH5
```

STEP Step in Block Mode

Format	STEP {PROG *number* | ALL}
Group	Program Control
Units	None
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	AUT, BLK, PAUSE, PROG, RESUME
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Executes the next line in block (BLK) mode.

Argument

PROG *number*	Program number 0 through 15
ALL	All programs currently in block mode

Remarks

The **STEP** command can be issued from anywhere in the system, including programs; however, block mode is not applicable to PLC programs.

The following is a list of valid **STEP** command combinations:

STEP PROG	Steps the corresponding program.
STEP ALL	Steps all programs.

When you use **STEP** in block mode, the program's "step request" bit is set and the next line of the current program executes.

Block mode and the **STEP** command can be used together in the debugging process. Use them to create an automatic breakpoint in a program during the run process, allowing closer examination of the subsequent steps.

Step requests are ignored under the following conditions:

- The program is not in block mode.
- The program is in pause mode.
- An attached master is executing a move but is not in feedhold.

If a move is in progress when the block mode is entered, the first **STEP** releases the feedhold on the active move and prevents the buffered move from executing by setting the master "start move inhibit" bit. The second **STEP** will then clear the master "start move inhibit" bit to start the buffered move. Any further **STEP** commands will operate normally.

If the **STEP** command starts a move that is then stopped with a master "feedhold request," the next **STEP** does not execute the next line of the program. Instead, it releases the feedhold so that the move in progress can complete.

To end block mode, issue the **AUT** command, then issue a **STEP** command or set the “step request” bit.

Example

The following illustrates where the **STEP** command is used in block mode:

```
PROGRAM
BLK : REM Program switches to block mode automatically
__
  \
   \
    > : REM A STEP command or step request bit is required at the program
   /           prompt in order to execute each line in this part of the
 _/            program.

AUT : REM Program switches to Autorun mode automatically.
ENDP
```

The following executes the next line in block mode:

```
BLK
STEP
```

In the following, the “step request” bit is coded into the program and the step will execute automatically:

```
P00>BLK
P00>SET 1045
P00>SET 1045
P00>AUT
P00>SET 1045
```

Example Bits

Flag Parameter 4128 For Program 0	
Step Request	1045

Flag Parameter 4112 For Master 0	
Feedhold Request	520
Start Move Inhibit	540

STP Set Stop Ramp

Format	STP {*rate*}
Group	Velocity Profile
Units	units/second^2 based on PPU
Data Type	FP32
Default	20000
Prompt Level	PROGx
Report Back	Yes **To view, type STP**
See Also	ACC, DEC, FVEL, IVEL, MASTER, PPU, VEL
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Sets the master deceleration for ramping to a stop at the end of coordinated motion.

Arguments

rate

Optional. Sets the stop rate for the master profile.

Remarks

Disabling Stop Ramp

With the *rate* set to zero, you can end a move without ramping down. This lets you merge a series of back-to-back moves. The velocity of an ending move becomes the initial velocity of the next move.

STP and FVEL

When the *rate* is greater than zero, the move ramps down to the final velocity rate set with **FVEL** command.

- If the **FVEL** rate is zero, the move comes to a complete stop.
- If the **FVEL** rate is greater than zero, the move ramps down to the final velocity rate at the end of the move. At the start of the next move, the

DEC versus STP

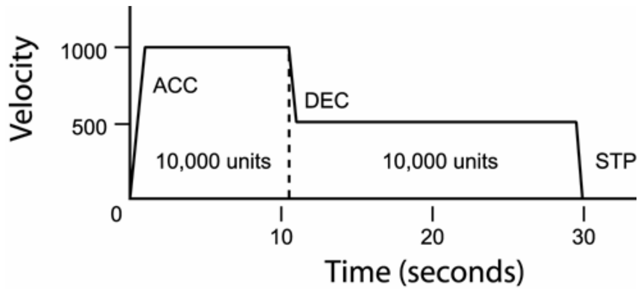
The table below summarizes the circumstances where **DEC** or **STP** are used. The **DEC** command can affect **CIRCCW**, **CIRCW**, **INT**, **MOV**, **PAUSE**, **SINE**, **STP**, and **TRJ**.

Description	DEC	STP
A Master's "Stop all Moves Request" bit is set. For example, Bit523 for Master 0.	Yes	No
A Master's "Feedhold Request" bit is set. For example, Bit520 for Master 0.	Yes	No
A program issues a **PAUSE**, which generates a "Feedhold Request" for the attached master.	Yes	No
A Master's **STP** rate is zero, and the next move segment's `VEL` rate is lower than the current segment's.	Yes	No
A Master's **FOV** rate is changed to a lower value during a move segment.	Yes	No
A Master's **ROV** rate is changed to a lower value during a move segment, and the "Rapid Active" bit is set.	Yes	No
When using **TRJ** moves.	Yes	No
Interpolated motion profile completes.	No	Yes

NOTE: For the ACR9000, hardware and software limits use the **HLDEC** and **SLDEC** for deceleration on contacting an end-of-travel limit. These rates override the **DEC** and **STP** values.

Example

The following example sets up a simple motion profile. The first move accelerates to velocity and moves 10,000 incremental units. At the start of the second move, the motor ramps down to the new velocity using the previously stated deceleration rate. At the end of the second move, the stop ramp ends motion.



```
DEC 1000 ACC 1000 STP0 VEL1000
X/10000   : REM move 10,000 incremental units
STP1000 VEL500
X/10000   : REM move 10,000 incremental units
```

STREAM Display Stream Name

Format	STREAM
Group	Operating system
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	N/A
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Displays the name of the current stream on which the command was issued.

Remarks

The following is a list of valid command combinations:

STREAM TIMEOUT	Set the timeout for TCP/IP communications when not using ACR-View.

Example

```
SYS>STREAM
COM1
SYS>
```

STREAM TIMEOUT Display Stream Name

Format	STREAM TIMEOUT *seconds*
Group	Operating system
Units	Seconds
Data Type	LONG
Default	0
Prompt Level	SYS, PROGx, PLCx
See Also	STREAM
Related Topics	N/A
Product Revision	Command and Firmware Release

The **STREAM TIMEOUT** command sets a connection timeout value for the current, TCP/IP command stream. This lets you set a watchdog to observe and manage timeouts for the TCP/IP connection. You cannot set a timeout value for non-TCP/IP connections (such as COM1 or USB).

NOTE: When using ACR-View, do not set a timeout value. The ACR-View software manages the connections for you. The **STREAM TIMEOUT** command is necessary only when providing your own communications server. For more information, see *ComACRsrvr User's Guide for ACR Series Controllers* (p/n 88-025359-01).

The default timeout value is zero, so no timeout can occur. The timeout value is not saved and has no corresponding parameter. The association between streams and connections is dynamic, and is made when the controller makes a TCP/IP connection. The association cannot persist, as the association with a specific stream or connection could result in an undesired disconnect from an application with a legitimate lack of activity. Therefore, you must set the timeout value each time you establish a connection.

Example

The following example queries for the current communications stream, then sets the stream timeout to 30 seconds.

```
SYS>STREAM
STREAM3
SYS>STREAM TIMEOUT 30
SYS>STREAM TIMEOUT
30
```

SYNC Synchronization of Masters

Format	SYNC {*command*}
Group	Velocity Profile
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	MASTER, PROG, SYNC PROG, SYNC MDI, TMOV VEL
Related Topics	Secondary Master Flags (0-7) (8-15)
Product Revision	Command and Firmware Release

This group of commands is used with or without the **TMOV** commands to synchronize the moves of the masters. Any number and combination of masters can be synchronized together. Using the synchronized moves, instead of coordinated motion, gives the flexibility of using different motion profiles for axes connected to different masters and still be in sync.

The following is a list of valid **SYNC** command combinations:

SYNC MDI: Synchronize moves from immediate mode.

SYNC OFF: Turn off synchronization of masters.

SYNC ON: Synchronize moves of masters.

SYNC PROG: Synchronize moves from programs.

Trying to sync a master that is already in sync with another group or has no master profile, will return the respective message. Since the **SYNC ON** command can't be given before attaching the masters, it is preferred to use this command from the last program in the sync group. Issuing just the **SYNC** command with no argument will show the masters that are in sync with the program/master from which the command is issued. If the masters in the sync group need to be changed, then first use the **SYNC OFF** command to cancel the group and then form a new sync group by using **SYNC ON** command.

If any master in a sync group is given a move command that cannot be done within the time specified by **TMOV**, then the masters in the sync group will automatically calculate the extra time and the profile that will be needed to slow down their moves to keep in sync. Extra time demanded by each master can be seen in the Master Parameter "Delta **TMOV** Time".

NOTE: The **SYNC** and **TMOV** commands are computationally intensive. Check the **CPU** load, and if the background **CPU** time is getting close to 60%, then increase the period. For example, to run more than 4 masters on ACR8000 at the same time, it is recommended to change the period to one millisecond.

There is also a limit on how short the move can be in time. This limit could be between 5 milliseconds to 100 milliseconds, depending on the system configuration.

Example 1

In this example, 4 masters are used. In the group, Master0, Master1, Master2, and Master3, are synced together. Each master makes a pattern of eight moves.

For convenience, 1st move is at line 10, 2nd move at line 20, 3rd move at line 30, and so on for each master)

```
PROG0
ATTACH MASTER0
ATTACH SLAVE0 AXIS0 "X"
TMOV ON

_LOOP1
ACC 1000 DEC 500 STP 0
VEL 300
X/200
X/200
X/200
X/200
X/200
X/200
X/200
STP1000
X/200
GOTO LOOP1

PROG2
ATTACH MASTER2
ATTACH SLAVE2 AXIS2 "Z"
TMOV 1
TMOV ON

_LOOP1
ACC 0 STP 0 VEL 300
Z/300
Z/300
ACC 1000 STP 1000 IVEL 1
Z/20
STP 0
Z/200
IVEL 0 STP 1000
Z/200
ACC 0 STP 0
Z/100
IVEL 1 ACC 1000
Z/200
IVEL 0 STP 1000
Z/200
GOTO LOOP1

PROG1
ATTACH MASTER1
ATTACH SLAVE1 AXIS1 "Y"
TMOV ON

_LOOP1
IVEL 0 FVEL 0
ACC 1200 DEC 0 STP 1200 VEL 300
```

```
Y/200
STP 0
Y/200
STP 1200
Y/200
Y/0
STP 0
Y/200
STP 1200
Y/200
STP 0
Y/200
STP1200
Y/200
GOTO LOOP1

PROG3
ATTACH MASTER3
ATTACH SLAVE3 AXIS3 "A"
SYNC ON MASTER0 MASTER1 MASTER2 MASTER3
TMOV ON

_LOOP1
ACC 100000 DEC 100000 STP 100000 VEL 500
TMOV .1
A 20
A 0
A 20
A 0
A 20
A 0
A 20
A 0
GOTO LOOP1
```

Example 2

In the following example, Master0 and Master1 each make their arcs in 3 seconds.

```
PROG0
ATTACH MASTER0
ATTACH SLAVE0 AXIS0 "X"
ATTACH SLAVE1 AXIS1 "Y"
TMOV 3
TMOV ON
_LOOP1
X0 Y0
X10000 Y0
SINE X(0,90,90,10000) SINE Y(10000,0,90,10000)
GOTO LOOP1

PROG1
ATTACH MASTER1
ATTACH SLAVE2 AXIS2 "Z"
ATTACH SLAVE3 AXIS3 "A"
TMOV 3
TMOV ON
SYNC ON MASTER0 MASTER1
_LOOP1
Z0 A0
Z5000 A0
SINE Z(0,90,90,5000) SINE A(5000,0,90,5000)
GOTO LOOP1
```

SYNC OFF Asynchronization of Masters

Format	SYNC OFF {*master*} {*master*} ...
Group	Velocity Profile
Units	None
Data Type	N/A
Default	OFF
Prompt Level	PROGx
See Also	MASTER
Related Topics	Secondary Master Flags (0-7) (8-15)
Product Revision	Command and Firmware Release

This command is used to take out any number and combination of masters from the sync mode. This command will release the master from the sync group and then the master will be able to independently execute its moves. This command can be issued any time, even if the masters are in motion.

Example

```
SYNC OFF MASTER0 MASTER2 MASTER3
```

SYNC ON Synchronization of Masters

Format	SYNC ON {*master*} {*master*} ...
Group	Velocity Profile
Units	None
Data Type	N/A
Default	OFF
Prompt Level	PROGx
See Also	MASTER
Related Topics	Secondary Master Flags (0-7) (8-15)
Product Revision	Command and Firmware Release

This command enables the synchronization of selected masters.

The **SYNC ON** command should be issued before the start of the moves. Issuing a Feed Hold Request/Cycle Start Request to one of the master in sync group will make all the masters in sync group to stop/start with their respective stop/acceleration ramps.

Example

```
SYNC ON MASTER0 MASTER1
```

SYNC MDI Synchronized Moves From Immediate Mode

Format	SYNC MDI
Group	Velocity Profile
Units	None
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	PROG
Related Topics	N/A
Product Revision	Command and Firmware Release

This command is used to tell the card that the moves will be issued manually from the immediate mode and not from program i.e. PROG0 through PROG7. This command should also be used when sending binary moves. This command will set the Secondary Master Flag “Sync Manual” of all the masters in sync.

In this mode, the masters start their sync moves as soon as all the masters in sync are loaded with their respective moves.

Example

```
SYNC MDI
```

SYNC PROG Synchronized Moves From Program Mode

Format	SYNC PROG
Group	Velocity Profile
Units	None
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	MASTER, PROG
Related Topics	N/A
Product Revision	Command and Firmware Release

Issuing this command tells the controller that the sync moves will be coming from the programs. This is the default for the sync mode. This command will clear the Secondary Master Flag “Sync Manual”.

Example

```
SYNC PROG
```

Program Example

It is assumed that two masters and axes are attached and are in sync mode prior to issuing the following commands

Sequence of Commands		Comments
PXX>	SYNC MDI	Issued once to tell that the moves will be issued from immediate mode
P00>	X/100	Loads the X-axis move
P01>	Y/200	Loads the Y-axis move, since both moves have been loaded, the two masters start the sync move.
P01>	Y/55	Load move
P00>	X 500	Load move and starts the moves
:	:	:
:	:	:
PXX>	SYNC PROG	Puts back the default sync mode in which the moves will be issued from the programs

NOTE: The above two commands, **SYNC PROG** and **SYNC MDI**, are only valid when the master is in **SYNC** mode.

SYS Return to System Prompt

Format	SYS
Group	Operating System
Units	None
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	ATTACH, AXIS, DIM, PLC, PROG
Related Topics	N/A
Product Revision	Command and Firmware Release

This command activates the system prompt. When a communications channel is activated, it is at the system level by default. Only limited commands can be executed from this level. The prompt at this level is as follows:

```
    SYS>_
```

The communications channel must be at a program level in order to edit and run programs. This is done via the **PROG** command. The **SYS** command cannot be executed from within a program.

Example

```
SYS
```

TALK TO Talk to Device

Format	TALK TO {#*device*}
Group	Character I/O
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	CLOSE, DTALK, OPEN
Related Topics	N/A
Product Revision	Command and Firmware Release

The **TALK TO** command lets you open manual communications (by means of a COM1, COM2, or STREAMn). Before you can use the **TALK TO** command, you must open communication with a device—use the **OPEN** command. In addition, you can use the **CLOSE** command to end communications with the device.

NOTE: The **TALK TO** command lets you communicate directly to another device, allowing commands to pass through the controller. If you are using controllers and drives that have the Drive Talk feature, use the **DTALK** command instead.

TANG Tangential Axis

Format	TANG *command* {*data*}
Group	Interpolation
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	MOV, NURB, SINE, SPLINE, TARC
Related Topics	Master Parameters (0-7) (8-15)
Product Revision	Command and Firmware Release

The **TANG** command is useful for putting an axis to an angle to a motion path, like perpendicular, tangential or any user set angle. When an axis is put in **TANG** mode, it will automatically move as the motion path changes. Keeping itself to a set angle with the motion path.

TANG OFF Turn Off the Tangential Axis

Format	TANG OFF
Group	Interpolation
Units	None
Data Type	N/A
Default	OFF
Prompt Level	PROGx
See Also	MOV, NURB, SINE, SPLINE, TARC
Related Topics	Secondary Master Flags (0-7) (8-15)
Product Revision	Command and Firmware Release

The **TANG OFF** command turns off the three dimensional circular interpolation mode for the master.

Example

```
PROG0>TANG OFF
```

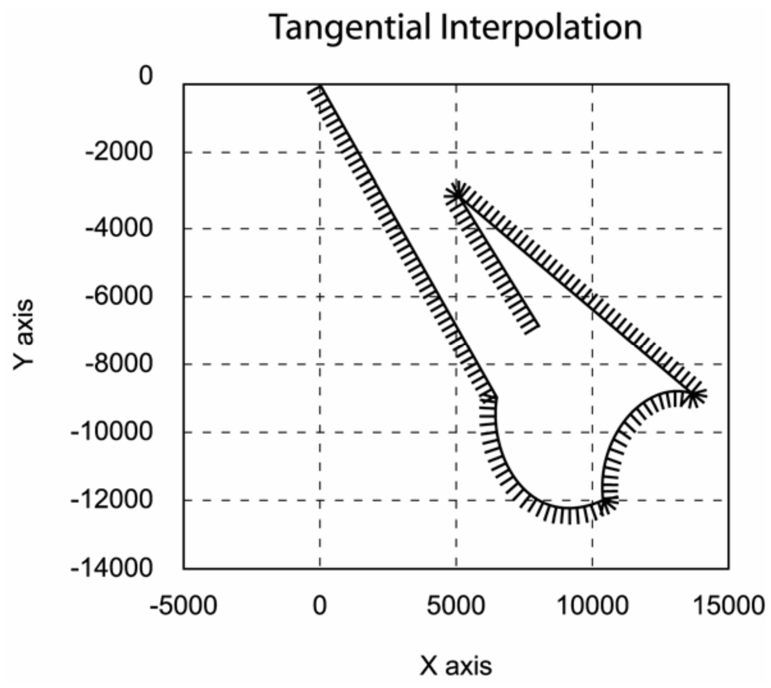
TANG ON Locks an Axis to a Motion Path With Any Angle

Format	TANG ON {*axis1 axis2 axis3*} ANG *angle*
Group	Interpolation
Units	Angle=degrees
Data Type	FP32
Default	OFF
Prompt Level	PROGx
See Also	MOV, NURB, SINE, SPLINE, TARC
Related Topics	Secondary Master Flags (0-7) (8-15)
Product Revision	Command and Firmware Release

The **TANG ON** command is used to put axis 1 tangential to the path traced by axis 2 and axis 3. Appending this command with **ANG** command, one can keep to any angle to the motion path of the two axes.

When the two adjacent moves have discontinuity in angle, then a move is automatically inserted to smoothly rotate the tangential axis. The user can separately set the acceleration and velocity of these inserted moves. The user can also specify minimum discontinuity between the moves for inserting the move. This is done by master parameter "TANG Turn Limit". (Version 1.18.06 update 09).

The following illustrates tangential interpolation:



Example

The following example will put z-axis perpendicular to motion of X and Y-axis. See the figure for detail.

```
TANG ON Z X Y ANG 90
X638 Y -907
CIRCCW X(1042.48, 916) Y(-1200.79,-950)
CIRCW X(1364.77, 1332) Y(-889.156,-1180)
X 500 Y -312.26
X800 Y -700
```

TARC 3-D Arc

Format	TARC {*command*}
Group	Interpolation
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	PROGx
Report Back	Yes **To view, type** **TARC**
See Also	MOV, NURB, SINE, SPLINE, TANG
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Generates a three-dimensional arc.

Arguments

command

Optional. Uses an additional daughter command. See *Associated Commands*, below, for a list of related commands.

Remarks

You can trace any three-dimensional arc between 0 to 360 degree using the three-dimensional circular interpolation mode. Each arc uses start, intermediate, and end points to define its placement in three-dimensional space. You can specify incremental or absolute moves for the defining points.

Points	Description
Start	The current position of the axes, when you turn on **TARC**, is the starting point for the first arc.
Intermediate	The intermediate point determines the direction and rotation of an arc.
End	The end position of an arc is the starting position for the next arc.

When you turn on **TARC**, the coordinated motion profiler uses the current actual position as the starting point of the first arc. You provide the intermediate and end points to define the arc. For subsequent arcs, the end point of an arc is the starting point for the subsequent arc.

If the starting, intermediate, and ending points lie in a straight line, then the coordinated motion profiler performs linear interpolation.

NOTE: The axes used with **TARC** must use the same PPU (pulses per programming unit); otherwise, you can receive unpredictable results.

Associated Commands

TARC OFF: Turns three-dimensional arcs off.

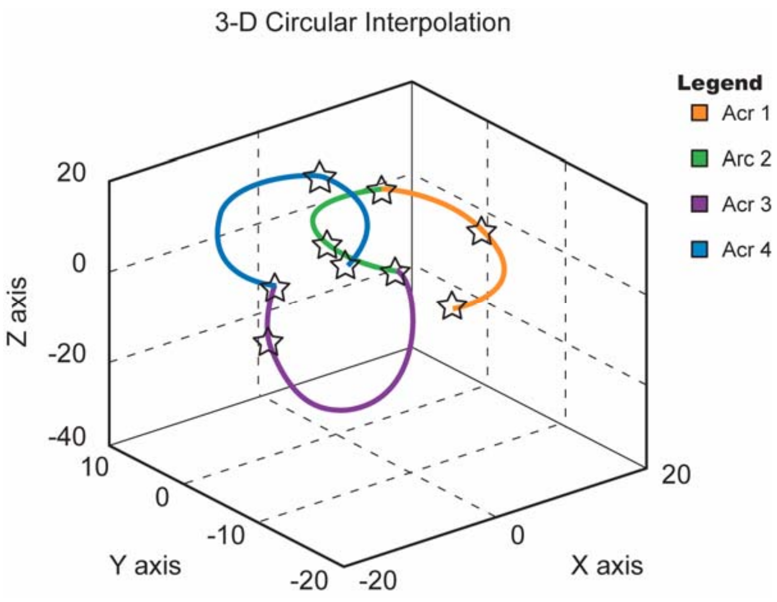
TARC ON: Turns three-dimensional arcs on.

Example

The following example moves to the coordinates X1 Y-14 Z1, and then turns on **TARC** mode.

```
TARC OFF
X1 Y-14 Z1    : REM moves to starting position
TARC ON X Y Z    : REM turns on 3D arc mode
REM XYZ-axes are put to 3-D circular interpolation mode
X13 Y-6 Z5    : REM 1st Arc intermediate point
X8 Y2 Z10    : REM 1st Arc end point
X5 Y6 Z-5    : REM 2nd Arc
X10 Y2 Z-10 A2
REM A-axis will do linear coordinated move with XYZ-axes
X0 Y0 Z-4.5    : REM 3rd Arc intermediate point
X-10 Y-2 Z1    : REM 3rd Arc end point
X1 Y3 Z15    : REM 4th Arc
X-1 Y-2 Z1
```

The following illustrates 3-D Arc interpolation:



TARC OFF Turn Off 3-Dimensional Circular Interpolation Mode

Format	TARC OFF		
Group	Interpolation		
Units	None		
Data Type	N/A		
Default	OFF		
Prompt Level	PROGx		
Report Back	Yes	**To view, type**	**TARC**
See Also	MOV, NURB, SINE, SPLINE, TANG, TARC		
Related Topics	Secondary Master Flags (0-7) (8-15)		
Product Revision	Command and Firmware Release		

Description

Turns off three-dimensional, circular interpolation mode.

Arguments

None

Remarks

The **TARC OFF** command turns off the three-dimensional circular interpolation mode.

Example

```
TARC OFF
```

TARC ON 3-Dimensional Circular Interpolation Mode

Format	TARC ON {*axis axis axis*}		
Group	Interpolation		
Units	NONE		
Data Type	N/A		
Default	OFF		
Prompt Level	PROGx		
Report Back	Yes	**To view, type**	**TARC**
See Also	MOV, NURB, SINE, SPLINE, TANG, TARC		
Related Topics	Secondary Master Flags (0-7) (8-15)		
Product Revision	Command and Firmware Release		

Description

Turns on three-dimensional, circular interpolation mode.

Arguments

axis

Optional. Assigns an axis.

Remarks

The **TARC ON** command turns on the three-dimensional circular interpolation mode.

Example

```
TARC ON X Y Z
```

TLM Set Torque Limit

Format	TLM {*axis* {*value*}} {*axis* {(*high*, low)}} ...
Group	Axis Limits
Units	Volts
Data Type	FP32
Default	10,-10
Prompt Level	PROGx
See Also	AXIS, EXC, ITB, JLM
Related Topics	Axis Flags (0-7)(8-15), Axis Parameters (0-7) (8-15), Master Flags (0-7) (8-15)
Product Revision	Command and Firmware Release

This command sets the voltage limits monitored by the "not torque limit" flags. When the output voltage of a given axis is not being torque limited, the appropriate flag is set. Otherwise, the flag is cleared. For masters, the flag is set if none of its slaves are being torque limited.

The limits set by the **TLM** command cause the output of the servo loop to be clipped at the given values. See the ITB command for non-clipping torque monitoring.

Issuing the **TLM** command to an axis without an argument displays the current positive and negative limits for that axis. Issuing the command with a single argument sets the positive limit to "value" and the negative limit to minus "value". Issuing the command with two arguments sets the positive limit to "high" and the negative limit to "low". The default values are ±10.0 volts for all axes.

Example 1

The following example sets the torque limit to ±1.5 volts for both X and Y axes.

```
TLM X1.5 Y1.5
```

Example 2

The following example sets the torque limit to 5 volts and –2 volts.

```
TLM X(5,-2)
```

TMOV Time Based Move

Format	TMOV {*time_in_seconds*}
Group	Velocity Profile
Units	Seconds
Data Type	FP32
Default	1
Prompt Level	PROGx
See Also	MOD, SYNC, TMOV OFF, TMOV ON, TOV
Related Topics	Master Parameters (0-7) (8-15), Secondary Master Flags (0-7) (8-15)
Product Revision	Command and Firmware Release

This commands sets the time in seconds in which the moves will be completed. Issuing a **TMOV** command without an argument will display the current value.

The following is a list of valid **TMOV** command combinations:

TMOV: Set time-based move time.

TMOV OFF: Disables time-based move.

TMOV ON: Activates time-based move.

TMOV VEL: Set synchronized master speed.

This command automatically calculates the new master profile parameters to do the moves in the specified time. The new parameters, i.e. **ACC**, **DEC**, **STP** and **VEL**, are not greater than the user specified value. In other words, the user can specify the boundary of the master profile, and the **TMOV** commands will remain within this boundary. If the time specified is too short to complete the move, then a Secondary Master Flag bit, "Master Short Time", is set, to indicate that the **ACC**, **DEC**, **STP** and **VEL** limits are hit. In this case, the move is carried out using the limit values.

The **TMOV** command only becomes active when it has been turned on by the **TMOV ON** command. It will remain on, unless the user turns it off by issuing a **TMOV OFF** command.

When using the **TMOV** command, use the same values for **ACC**, **DEC** and **STP**, if other than zero. Any combination of initial and final velocities can be used to make a move. If the user enters the initial and final velocity for the move, then these values will override the internal velocity profiler values. However, the user should not enter **FVEL** (final velocity) greater than (Move Distance/Move Time).

NOTE: **FOV** and **ROV** commands issued to the master in motion will make it slip in time for the current and the next move in the buffer. However, the subsequent moves, yet to be calculated, will adjust for this time slip. Instead, the **TMOV** command can be used on the fly to speedup or slow down, and the effect would be seen after the next move to be executed, which is already in the buffer.

Example 1

The following example sets the time for the moves to complete in 0.5 seconds.

```
TMOV 0.5
```

Example 2

The following example shows how to use the **TMOV** commands in a program.

```
PROG0
ATTACH MASTER0
ATTACH SLAVE0 AXIS0 "X"
ATTACH SLAVE1 AXIS1 "Y"
ACC 40000 DEC 40000 STP 40000 VEL 2000
_LOOP1
TMOV ON
TMOV .5
X100
X/400
TMOV 1
X/100
TMOV OFF
X1000
TMOV ON
X500
X0
GOTO LOOP1
```

TMOV OFF Disable Time Based Move

Format	TMOV OFF
Group	Velocity Profile
Units	None
Data Type	N/A
Default	OFF
Prompt Level	PROGx
See Also	MOV, SYNC, TOV
Related Topics	N/A
Product Revision	Command and Firmware Release

This command will disable the time based moves. The moves will be done by the user specified master profile parameters. This command will Clear the Secondary Master Flags "Master in TMOV" and "Master Short Time". Halting the program will also clear both the flags.

Example

The following example disables the time based moves.

```
TMOV OFF
```

TMOV ON Set Time Based Move

Format	TMOV ON
Group	Velocity Profile
Units	None
Data Type	N/A
Default	OFF
Prompt Level	PROGx
See Also	MOV, SYNC, TOV
Related Topics	N/A
Product Revision	Command and Firmware Release

This command will activate the time based moves. The moves will be done in the time set by the **TMOV** command. At the time this command is issued the current master profile parameters will be saved. This command will Set the Secondary Master Flag "Master in TMOV".

Example

The following example activates the time based moves.

```
TMOV ON
```

TMOV VEL Change the Speed of a Master in Sync Mode

Format	TMOV VEL {*value_of_velocity*}
Group	Velocity Profile
Units	Units scalable by PPU
Data Type	FP32
Default	1
Prompt Level	PROGx
See Also	FOV, MOV, SYNC, TMOV, VEL
Related Topics	N/A
Product Revision	Command and Firmware Release

In sync mode, this command can be used to change the speed. However, the effect will be seen after the next buffered move. Using this command instead of **FOV** will avoid the jerk/slip in time.

Example

The following example sets the sync move speed to 3000.

```
TMOV VEL 3000
```

TOV Time Over Velocity

Format	TOV {*value*}
Group	Velocity Profile
Units	None
Data Type	FP32
Default	1
Prompt Level	PROGx
See Also	FOV, SYNC, TMOV OFF, TMOV ON
Related Topics	Master Parameters (0-7) (8-15)
Product Revision	Command and Firmware Release

This command is used for immediately changing the speed of the masters in motion. For example, changing the **TOV** from 1 to 2 will make the moves happen in half the time, or double the speed. Issuing this command will affect the velocity and acceleration as follows:

New VEL = **VEL** ***TOV**

New ACC = **ACC*****TOV*****TOV**

This change is not sudden, and the user can tune the rate of change, so that the motion profile is smooth. The rate of change for each master can be adjusted by setting the Master Parameter "TOV RATE" (default value is 2). Keep this value the same, for all the masters in sync.

Issuing this command with no argument will display the current setting.

NOTE: Issuing the **TOV** command for the first time attaches a special source to the master profiler. So the first time this command may be issued is when the master is not moving, otherwise a very small glitch might be seen.

NOTE: If the user decides not to use **TOV** command, then it should be turned off by clearing the Master Secondary Flag "Master TOV". This will reduce unnecessary load on the **CPU**.

NOTE: **TOV** command stretches or compresses the time source. Therefore, it will be suitable for making small variations to immediately change the speed.

Example

```
P00>TOV
P00>1
P00> TOV 2     : REM This doubles the speed
```

Why TOV and not FOV

TOV is used where **FOV** will not work. For example in non-linear motion profiles like spline, NURB and **SYNC**.

TRG Start Move on Trigger

Format 1:	TRG + *index*
Format 2:	TRG - *index*
Group	Logic Function
Units	None
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	CLR, INH, SET
Related Topics	Master Flags (0-7) (8-15)
Product Revision	Command and Firmware Release

This command stores the given bit index and sets a master flag that indicates a pending trigger operation. The next move will inhibit on this condition and start immediately after the condition becomes true. This reduces the amount of time between an inhibit and the start of a move since the information is processed before the condition is met.

The program in which the **TRG** command was executed will pause execution at the commanded move line until the bit in the **TRG** command condition is met. All code between the **TRG** and commanded move line is executed.

The "+ index" requires the bit to be set (-1) to initiate motion. The "- index" requires the bit be cleared (0) to initiate motion.

Example

The following example will preload the move in line 50 and start the move as soon as bit 7 (INP07) becomes active. The program will turn on output 32 and reset the position for both axes X and Y to zero. The program execution then pauses at line 50 until the condition in line 20 is met. It then waits until both axes X and Y have finished moving. Output 32 is then turned off in line 80. The program next preloads the move in line 110 and starts the move when bit 128 is cleared. Before and after the move, the program prints the actual position for axes 0 and 1.

```
ACC100 DEC100 ST100 VEL1
TRG+7
SET32
RES X Y
X400 Y400
WHILE (BIT 516=-1 OR BIT 517=-1)
WEND
CLR32
TRG-128
PRINT P12290, P12546
X0 Y0
WHILE (BIT 516=-1 OR BIT 517=-1)
WEND
PRINT P12290, P12546
```

TRJ Start New Trajectory

Format	TRJ {*axis target*} {*axis target*} ...
Group	Interpolation
Units	Units based on PPU
Data Type	None
Default	N/A
Prompt Level	PROGx
Report Back	None **To view, type** N/A
See Also	MOV, SINE, PPU
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Changes the trajectory of axes on-the-fly.

Arguments

axis

Optional. Assigns an axis.

target

Optional. Position at the end of the move (in units).

Remarks

When the **TRJ** command is executed, the axes continue along the specified vector until one of the following conditions are met:

- The move completes normally.
- The move is aborted (**HALT** (or ESC in MDI)).
- The coordinated motion profiler receives another **TRJ** command. Where commanded motion with **MOV** is buffered, **TRJ** moves can change vector with minimal delay.

NOTE: The **TRJ** command has 50 millisecond execution time.

Example

The following example commands the X and Y axes to move along a 45 degree vector, and after 10 seconds move at a 90 degree vector.

```
TRJ X100000 Y100000
DWL 10
TRJ Y100000
```

TROFF Turn Off Trace Mode

Format	TROFF {PROG *number* | ALL}
Group	Program Control
Units	N/A
Data Type	N/A
Default	OFF
Prompt Level	SYS, PROGx, PLCx
See Also	AUT, BLK, HALT, LISTEN, LRUN, PAUSE, RESUME, RUN, TRON
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Turns off tracing for the currently selected program.

Argument

PROG *number* | ALL

Remarks

When tracing is turned on, the program displays the line number of each line as it is executed. The displayed line number is enclosed in angle brackets.

The **TROFF PROG** command turns off tracing for the corresponding program. The **TROFF ALL** command turns off tracing for all programs. These commands can be issued from anywhere in the system, including programs. To see the trace, use the LRUN command.

NOTE: Trace mode does not slow down program execution. Communication can lock up due to the amount of information being written out.

Example

The following shows a program turning on and off its own tracing:

```
SYS>PROG0
P00>NEW
P00>10 PRINT "A";
P00>20 PRINT "B";
P00>30 TRON
P00>40 PRINT "C";
P00>50 PRINT "X";
P00>60 TROFF
P00>70 PRINT "Y";
P00>80 PRINT "Z";
P00>90 PRINT
P00>LRUN
AB<40>C<50>XYZ
P00>_
```

TRON Turn On Trace Mode

Format	TRON {PROG *number* | ALL}
Group	Program Control
Units	N/A
Data Type	N/A
Default	OFF
Prompt Level	SYS, PROGx, PLCx
See Also	AUT, BLK, HALT, LISTEN, LRUN, PAUSE, RESUME, RUN, TROFF
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Turns on tracing for the currently selected program when in **LRUN** or **LISTEN** mode.

Argument

PROG *number* | ALL

Remarks

When tracing is turned on, the program displays the line number of each line as it is executed. The displayed line number is enclosed in angle brackets.

The **TRON PROG** command turns on tracing for the corresponding program. The **TRON ALL** command turns on tracing for all programs. These commands can be issued from anywhere in the system, including programs. To see the trace, use the LRUN or LISTEN commands.

NOTE: Trace mode does not slow down program execution. Communication can lock up due to the amount of information being written out.

Example

The following shows a program running with and without tracing:

```
SYS>PROG0
P00>NEW
P00>10 DIM LV1
P00>20 LV0 = 0
P00>30 PRINT CHR$(65+LV0);
P00>40 LV0 = LV0+1
P00>50 IF (LV0 < 3) THEN GOTO 30
P00>60 PRINT
P00>TRON
P00>LRUN
<10><20><30>A<40><50><30>B<40><50><30>C<40><50><60>
P00>TROFF
P00>LRUN
ABC
P00>_
```

UNLOCK Unlock Gantry Axis

Format	UNLOCK {*axis*} {*axis*} ...
Group	Setpoint Control
Units	None
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	BSC, CAM, GEAR, HDW, JOG, LOCK
Related Topics	N/A
Product Revision	Command and Firmware Release

This command releases primary setpoint redirection that may have been established with the **LOCK** command. The actual position parameter of the axis is adjusted such that there is no change in following error when the primary setpoint switches. The default state of an axis is to follow its own setpoint.

Each axis generates a primary setpoint based on its current position, gear offset, jog offset, and cam offset. This number is normally used to tell the axis where it should be at any given time. The **LOCK** command tells an axis to use the primary setpoint of a different axis instead of its own. The **UNLOCK** command tells an axis to use its own primary setpoint once again.

Example

The following example releases axis Y from its primary setpoint redirection:

```
P12376 = 3.5
P12632 = 3.5
LOCK Y X
X /20
UNLOCK Y
```

VECDEF Define Automatic Vector

Format	VECDEF {*axis* {*weight*}} {*axis* {*weight*}} ...
Group	Velocity Profile
Units	None
Data Type	FP32
Default	1
Prompt Level	PROGx
See Also	MASTER, MOV, VECTOR
Related Topics	N/A
Product Revision	Command and Firmware Release

This command controls how the master move vector is calculated. The argument passed to an axis determines how much the axis contributes to the vector calculation. Issuing a **VECDEF** command to an axis without an argument will display the current setting for that axis. The default value is 1 for all axes.

With automatic vector calculation, the length of the master move is calculated using the following formula:

vector distance = sqrt (∑ (delta(n)^2 * weight(n)))

where:

n = axis index number (from 0..7)

delta(n) = distance axis(n) is moving

weight(n) = vector contribution of axis(n)

The master will internally move from zero to the vector distance, using the current **VEL**, **ACC**, **DEC**, **STP** and **FOV** settings to control its velocity profile. The axes attached to the master will start when the master starts and reach their target positions as the master finishes its move.

In many multi-axis configurations, it is not necessary (nor desirable) to have all the axes contributing to this calculation. For example, in a configuration containing three Cartesian axes and a rotary axes, just the Cartesian axes need to be included in the vector distance. Also, in single master / single slave setups, where the master distance is always equal to the slave distance, the default weight of 1.0 should be left alone.

For non-contributing axes, the vector weight should be set to zero. If these axes are to be moved by themselves, the automatic vector calculation must be overridden by using the **VECTOR** command. This may also require other initialization in preparation of the independent move.

Example

The following example makes an X,Y move with A axis interpolation. They all stop at the same time. The **VECTOR** command is then used to move the A axis by itself as if X and Y were moving along a vector that is 1200 units in length:

```
VECDEF X1 Y1 Z1 A0
X10000 Y20000 A270
VECTOR 1200
A0
VECTOR 0
```

VECTOR Set Manual Vector

Format	VECTOR {*length*}
Group	Velocity Profile
Units	Units based on PPU
Data Type	FP32
Default	0
Prompt Level	PROGx
See Also	PPU, VECDEF
Related Topics	Master Parameters (0-7) (8-15)
Product Revision	Command and Firmware Release

This command allows manual override of the default vector length calculations for moves. If **VECTOR** is set to zero, the vector length is calculated automatically as described in the **VECDEF** command. Issuing a **VECTOR** command without an argument will display the current setting. The default value is 0.

When a move is executed, the master controlling the move is actually making an imaginary move for a certain number of "units". The velocity profile of this move is controlled by the current **VEL**, **ACC**, **DEC**, **STP** and **FOV** settings.

The length of the master's move is called the "vector distance" and is either calculated automatically based on **VECDEF** and slave distances, or overridden manually as described here. The attached slaves start when the master starts and arrive at their target positions when the master finishes its move.

Example

The following example makes an X,Y move with A axis interpolation. They all stop at the same time. The **VECTOR** command is then used to move the A axis by itself as if X and Y were moving along a vector that is 1200 units in length:

```
VECDEF X1 Y1 Z1 A0
X10000 Y20000 A270
VECTOR 1200
A0
VECTOR 0
```

VEL Set Target Velocity for a Move

Format	VEL {*rate*}
Group	Velocity Profile
Units	units/sec scaleable by PPU
Data Type	FP32
Default	10000 units/second
Prompt Level	PROGx
Report Back	Yes **To view, type** **VEL**
See Also	ACC, DEC, FOV, MASTER, PPU, STP
Related Topics	Master Parameters (0-7) (8-15)
Product Revision	Command and Firmware Release

Description

Sets the target velocity for subsequent moves

This programmed velocity is over-ridden by the **FOV** command. The velocity changes at a rate set by the **ACC**, **DEC** and **STP** commands.

Arguments

rate

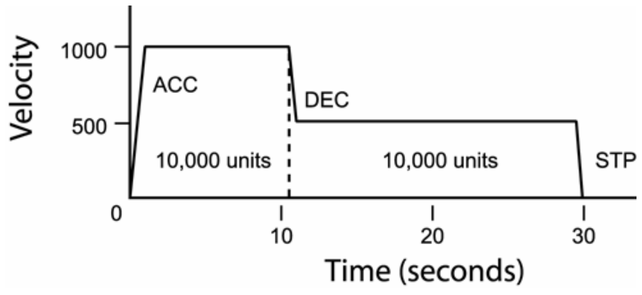
Optional. Sets the target velocity for the master profile.

Remarks

You can use the **FOV** command to immediately override the **VEL** rate. Changing the **VEL** rate, at a terminal emulator, does not change the rate of the move in progress. Only **FOV** can immediately override the **VEL** rate.

Example

The following example sets up a simple motion profile. The first move accelerates to velocity and moves 10,000 incremental units. At the start of the second move, the motor ramps down to the new velocity using the previously stated deceleration rate. At the end of the second move, the stop ramp ends motion.



```
DEC 1000 ACC 1000 STP0 VEL1000
X/10000    : REM move 10,000 incremental units
STP1000 VEL500
X/10000    : REM move 10,000 incremental units
```

VEL LIMIT Sets the Maximum Limit of Master Velocity

Format	VEL LIMIT {*value*}
Group	Velocity Profile
Units	Units/sec scaleable by PPU
Data Type	FP32
Default	0=no limit
Prompt Level	PROGx
See Also	FOV, MAXVEL, PPU, ROV, VEL
Related Topics	Quaternary Master Flags (0-7) (8-15), Master Parameters (0-7) (8-15)
Product Revision	Command and Firmware Release

This command is used to set the maximum velocity limit for a master profiler. This will protect against issuing a too high value of **VEL**, **FOV**, **ROV** commands etc

Example

```
VEL LIMIT 10
```

VER Display Board Type and Firmware Version

Format	VER
Group	Operating System
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	DIAG
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

Displays the board type and executive version.

Remarks

The board type will indicate whether the controller is an ACR1200, ACR1500, ACR1505, ACR2000 , ACR8000, ACR8010, ACR8020, or ACR9000. The **VER** command cannot be issued from within a program.

Example

```
SYS>VER
ACR1505(C) 2003 PARKER HANNIFIN CORPORTATION
VERSION 1.18.09 CARD 0
```

WHILE/WEND Loop Execution Conditional

Format	WHILE (*boolean*) commands WEND
Group	Program Flow
Units	N/A
Data Type	N/A
Default	N/A
Prompt Level	PROGx
See Also	BREAK, ENDP, IF THEN, PROGRAM
Related Topics	N/A
Product Revision	Command and Firmware Release

Description

This command loop is used to execute a set of statements if the Boolean expression is true. The **WHILE** loop is a pre-checked loop—the loop checks the conditional expression at the beginning of the loop and not the end. The **BREAK** command can be used to break out from the **WHILE** loop if certain other conditions are met.

NOTE: The **WHILE/WEND** statement does not have a built-in counter variable, and is vulnerable to infinite loops.

Examples

Example 1

```
#DEFINE lvCounter LV0
#DEFINE inStop BIT3

PROGRAM
DIM LV1
lvCounter = 0
WHILE (lvCounter < 10)
        PRINT "COUNTING "
        PRINT "SECONDS = ", lvCounter
        IF (inStop)
                PRINT " Stop Counting"
                BREAK
                ENDIF
        PRINT " Dwell for 1 second"
        DWL 1
        lvCounter = lvCounter +1
WEND
ENDP
```

Example 2

```
#DEFINE pTimeOut P2
#DEFINE inStop BIT3
#DEFINE SysClock P6916

PROGRAM
pTimeOut=5000    : REM set timeout to 5000 milliseconds
SysClock=0    : REM set system clock to zero
AXIS0 JOG FWD   : REM start a jog move
REM loop until system clock exceeds timeout value
```

```
WHILE (SysClock< pTimeOut)
       IF (inStop) THEN BREAK   : REM break out of loop if input 3
WEND
AXIS0 JOG OFF   : REM stop jog move
ENDP
```

Expression Reference

The AcroBASIC programming language includes mathematical expressions.

For more information about command formats, arguments, and syntax, see Command Syntax.

Expression Groups

Arithmetic	
Command	**Description**
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation
MOD	Modulus

Comparison	
Command	**Description**
<	Less than
<=	Less or equal
<>	Not equal
=	Equal to
>	Greater than
>=	Greater or equal

Hyperbolic	
Command	**Description**
ACOSH	Hyperbolic arc cosine
ACOTH	Hyperbolic arc cotangent
ASINH	Hyperbolic arc sine
ATANH	Hyperbolic arc tangent
COSH	Hyperbolic cosine
COTH	Hyperbolic cotangent
SINH	Hyperbolic sine
TANH	Hyperbolic tangent

Logical	
Command	**Description**
<<	Left shift
>>	Right shift
AND	Logical AND
BIT	Bit flag status
NAND	Logical NAND
NOR	Logical NOR
NOT	Logical NOT
OR	Logical OR
XNOR	Logical XNOR
XOR	Logical XOR

Miscellaneous	
Command	Description
ABSF	Absolute value of a number
CEIL	Smallest integer >= expression
FLOOR	Largest integer <= expression
LN	Natural logarithm
LOG	Common logarithm
RND	Random integer
ROUND	Round to nearest integer
SQRT	Square root
TRUNC	Remove fractional part

String	
Command	**Description**
ASC	ASCII Value
CHR$	Character string
GETCH	Wait for a character
INKEY$	Return a character
INSTR	String search
KBHIT	Check for waiting character
LCASE$	Convert to lower case
LEFT$	Left string
LEN	String length

String	
Command	**Description**
MID$	Middle string
RIGHT$	Right string
SPACE$	String of spaces
STR$	Convert numeric to string
STRING$	String of characters
UCASE$	Convert to upper case
VAL	Convert string to numeric

Trigonometric	
Command	**Description**
ACOS	Arc cosine
ACOT	Arc cotangent
ASIN	Arc sine
ATAN	Arc tangent
COS	Cosine
COT	Cotangent
SIN	Sine
TAN	Tangent

+ Addition

Format 1:	expression1 **+** expression2
Format 2	string_expression1 + string_expression2
Group	Arithmetic
Units	None
Data Type	FP32 or Long
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	–, *, /
Product Revision	Command and Firmware Release

This operator returns the value of expression1 plus expression2.

Example

```
P00>DIM LV2    : REM Dimension 2 Long Variables
P00>P6144=100    : REM Set P6144 equal to 100
P00>LV0=2    : REM Set the first Long Variable to 2
P00>LV1=P6144+LV0    : REM Set the second LV to the sum of P6144 and
LV0
P00>? LV1    : REM Print the value of LV1
102    : REM The result
```

-　　　　　Subtraction

Format	expression1 **-** expression2
Group	Arithmetic
Units	None
Data Type	Long
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	+, *, /
Product Revision	Command and Firmware Release

This operator returns the value of expression1 minus expression2.

Example

```
SYS>DIM P1000    : REM Dimension 1000 User Parameters
SYS>PROG0
P00>DIM SV2    : REM Dimension 2 Single Variables
P00>P999=100.1    : REM Set Parameter 999 to 100.1
P00>SV0=.02    : REM Set the first SV to 0.02
P00>SV1=P999-SV0    : REM Set second SV to difference between P999 and
SV0
P00>? SV1    : REM Print the value of SV1
100.08    : REM The result
```

*   Multiplication

Format	expression1 ***** expression2
Group	Arithmetic
Units	None
Data Type	FP32 or Long
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	+, –, /
Product Revision	Command and Firmware Release

This operator returns the value of expression1 multiplied by expression2.

Example

```
P00>P999=100    : REM Set Parameter 999 to 100
P00>SV0=.05    : REM Set the first Single Variable to 0.05
P00>? P999*SV0    : REM Print the result of P999 multiplied by SV0
5    : REM The result
```

/ Division

Format	expression1 **/** expression2
Group	Arithmetic
Units	None
Data Type	FP32 or Long
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	+, –, *
Product Revision	Command and Firmware Release

This operator returns the value of expression1 divided by expression2.

Example

```
P00>SV0=5    : REM Set the first Single Variable to 5
P00>SV1=.1    : REM Set the second Single Variable to 0.1
P00>P999=SV0/SV1    : REM Set Parameter 999 equal to SV0 divided by SV1
P00>PRINT P999    : REM Print the value of P999
50    : REM The result
```

** Exponentiation

Format	expression1 ** expression2
Group	Arithmetic
Units	None
Data Type	FP32 or Long
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	*
Product Revision	Command and Firmware Release

This operator returns the value of expression1 raised to the power of expression2.

Example

```
P00>SV0=2.0     : REM Set SV0 to 2.
P00>SV1=4.0     : REM Set SV1 to 4.
P00>PRINT SV0**SV1     : REM Print the result of SV0 to the exponential
of SV1.
16     : REM The result.
```

<<　　Left Shift

Format	expression1 **<<** expression2
Group	Logical
Units	None
Data Type	Long
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	>>
Product Revision	Command and Firmware Release

This operator returns the integer value of expression1 logically shifted to the left by expression2.

Example

```
P00>PRINT 1<<4
REM Print the result of 1 = Binary 00001 shifted Left by 4 places.
16    : REM The result is 16 = Binary 10000
P00>SV0=1.1    : REM Set SV0 to 1.1.
P00>? SV0<<2    : REM Print result of SV0 shifted Left by 2 places.
4    : REM The result. Note that the .1 was truncated.
```

>> Right Shift

Format	expression1 **>>** expression2
Group	Logical
Units	None
Data Type	Long
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	<<
Product Revision	Command and Firmware Release

This operator returns the integer value of expression1 logically shifted to the right by expression2.

Example

```
P00>P999=256    : REM Set P999 equal to 256 = Binary 100000000
P00>SV0=4    : REM Set SV0 equal to 4.
P00>SV1=P999>>SV0    : REM Set SV1 to result of P999 shifted Right by
REM SV0 places.
P00>? SV1    : REM Print the result stored in SV1.
16    : REM 16 = Binary 000010000
```

< Less Than

Format 1:	expression1 **<** expression2
Format 2:	string_expression1 **<** string_expression2
Group	Comparison
Units	None
Data Type	Boolean return
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	>, =, <>, <, >=
Product Revision	Command and Firmware Release

This operator returns -1 if the value of expression1 is less than expression2, otherwise it returns 0.

Example

```
P00>P900=1    : REM Set Parameter 900 equal to 1.
P00>SV0=.5    : REM Set Single Variable 0 equal to 0.5.
P00>SV1=2    : REM Set Single Variable 1 equal to 2.
P00>? P900<SV0
REM Print the result of whether P900 is less : REM than SV0 or not.
0    : REM The answer is False.
P00>PRINT P900<SV1
REM Print the result of whether P900 is less : REM than SV1 or not.
-1    : REM The answer is True.
P00>P899=P900<P900
REM Set Parameter 899 to the result of whether : REM P900 is less than
REM itself or not.
P00>? P899    : REM Print the value of P899.
0    : REM The value is zero...the result was False.
```

= Equal To

Format 1:	(expression1 **=** expression2)
Format 2:	(string_expression1 **=** string_expression2)
Group	Comparison
Units	None
Data Type	Boolean return
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	>, <>, <, >=
Product Revision	Command and Firmware Release

This operator returns -1 if the value of expression1 is equal to expression2, otherwise it returns 0.

NOTE: Parentheses are required to set the equivalence of the two expressions.

Example

```
P00>P900=1     : REM Set Parameter 900 equal to 1.
P00>SV0=.5     : REM Set Single Variable 0 equal to 0.5.
P00>SV1=2    : REM Set Single Variable 1 equal to 2.
P00>? (P900=SV0)
REM Print the result of whether P900 is equal to SV0 or not.
0     : REM The answer is False.
P00>PRINT (P900=SV1)
REM Print the result of whether P900 is equal to SV1 or not.
0     : REM The answer is False.
P00>P899=(P900=P900)
REM Set Parameter 899 to the result of whether P900 is equal to
REM itself or not.
P00>? P899     : REM Print the value of P899.
-1     : REM The value is -1...the result was True.
```

> Greater Than

Format 1:	expression1 **>** expression2
Format 2:	string_expression1 **>** string_expression2
Group	Comparison
Units	None
Data Type	Boolean return
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	=, <>, <, >=
Product Revision	Command and Firmware Release

This operator returns -1 if the value of expression1 is greater than expression2, otherwise it returns 0.

Example

```
P00>P900=1    : REM Set P900 equal to 1.
P00>SV0=.5    : REM Set SV0 equal to 0.5.
P00>SV1=2    : REM Set SV1 equal to 2.
P00>? P900>SV0
REM Print the result of whether P900 is greater than SV0 or not.
-1    : REM The result is True.
P00>PRINT P900>SV1
REM Print the result of whether P900 is greater than SV1 or not.
0    : REM The result is False.
P00>P899=P900>P900
REM Set P899 to the result of whether P900 is greater than itself.
P00>? P899    : REM Print the result.
0    : REM The result is False.
```

<> Not Equal To

Format 1:	expression1 **< >** expression2
Format 2:	string_expression1 **< >** string_expression2
Group	Comparison
Units	None
Data Type	Boolean return
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	>, =, <, >=
Product Revision	Command and Firmware Release

This operator returns -1 if the value of expression1 is not equal to expression2, otherwise it returns 0.

Example

```
P00>P900=1    : REM Set P900 equal to 1.
P00>SV0=.5    : REM Set SV0 equal to 0.5.
P00>SV1=2    : REM Set SV1 equal to 2.
P00>? P900<>SV0
REM Print the result of whether P900 is not equal to SV0 or not.
-1    : REM The result is True.
P00>PRINT P900<>SV1
REM Print the result of whether P900 is not equal to SV1 or not.
-1    : REM The result is True.
P00>P899=P900<>P900
REM Set P899 to the result of whether P900 is not equal to itself.
P00>? P899    : REM Print the result.
0    : REM The result is False.
```

<= Less Than or Equal

Format 1:	expression1 **< =** expression2
Format 2:	string_expression1 **< =** string_expression2
Group	Comparison
Units	None
Data Type	Boolean return
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	>, =, < >, <, > =
Product Revision	Command and Firmware Release

This operator returns -1 if the value of expression1 is less than or equal to expression2, otherwise it returns 0.

Example

```
P00>P900=1    : REM Set P900 equal to 1.
P00>SV0=.5    : REM Set SV0 equal to 0.5.
P00>SV1=2    : REM Set SV1 equal to 2.
P00>? P900<=SV0
REM Print result of whether P900 is less than or equal to SV0 or not.
0    : REM The result is False.
P00>PRINT P900<=SV1
REM Print result of whether P900 is less than or equal to SV1 or not.
-1    : REM The result is True.
P00>P899=P900<=P900
REM Set P899 to the result of whether P900 is less than
REM or equal to itself.
P00>? P899    : REM Print the result.
-1    : REM The result is True.
```

>= Greater Than or Equal

Format 1:	expression1 **>=** expression2
Format 2:	string_expression1 **>=** string_expression2
Group	Comparison
Units	None
Data Type	Boolean return
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	>, =, <>, <
Product Revision	Command and Firmware Release

This operator returns -1 if the value of expression1 is greater than or equal to expression2, otherwise it returns 0.

Example

```
P00>P900=1    : REM Set P900 equal to 1.
P00>SV0=.5    : REM Set SV0 equal to 0.5.
P00>SV1=2    : REM Set SV1 equal to 2.
P00>? P900>=SV0
REM Print result of whether P900 is less than or equal to SV0 or not.
-1    : REM The result is True.
P00>PRINT P900>=SV1
REM Print result of whether P900 is less than or equal to SV1 or not.
0    : REM The result is False.
P00>P899=P900>=P900
REM Set P899 to the result of whether P900 is less than
REM or equal to itself.
P00>? P899    : REM Print the result.
-1    : REM The result is True.
```

ABSF Absolute Value of a Number

Format	**ABSF** (*expression*)
Group	Miscellaneous
Units	N/A
Data Type	Float or Long
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	ROUND, RND
Product Revision	Command and Firmware Release

The **ABSF** function returns the absolute values of an expression.

Example

```
PRINT ABSF(1.25)
PRINT ABSF(-1.25)
PRINT ABSF(-175)
```

Example Output

```
1.25
1.25
175
```

ACOS Arc Cosine

Format	**ACOS** (*expression*)
Group	Trigonometric
Units	None
Data Type	FP32 or Long
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	SIN, COS, TAN, COT, ASIN, ATAN, ACOT
Product Revision	Command and Firmware Release

This function returns the arc cosine of the expression.

ACOSH Hyperbolic Arc Cosine

Format	**ACOSH** (*expression*)
Group	Hyperbolic
Units	None
Data Type	FP32 or Long
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	SINH, COSH, TANH, COTH, ASINH, ATANH, ACOTH
Product Revision	Command and Firmware Release

This function returns the hyperbolic arc cosine of the expression.

ACOT Arc Cotangent

Format	**ACOT** (*expression*)
Group	Trigonometric
Units	None
Data Type	FP32 or Long
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	SIN, COS, TAN, COT, ASIN, ACOS, ATAN
Product Revision	Command and Firmware Release

This function returns the arc cotangent of the expression.

ACOTH Hyperbolic Arc Cotangent

Format	**ACOTH** (*expression*)
Group	Hyperbolic
Units	None
Data Type	FP32 or Long
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	SINH, COSH, TANH, COTH, ASINH, ATANH, ACOTH, ACOSH
Product Revision	Command and Firmware Release

This function returns the hyperbolic arc cotangent of the expression.

AND Logical AND

Format	expression1 **AND** expression2
Group	Logical
Units	None
Data Type	Long
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	NAND, OR, NOR, XOR, XNOR, NOT, BIT
Product Revision	Command and Firmware Release

This operator returns the logical **AND** of the two expressions. Bits in the result will be set if the corresponding expression bits are both set.

Example

```
PRINT 0 AND 0
PRINT 0 AND -1
PRINT -1 AND 0
PRINT -1 AND -1
```

Example Output

```
0
0
0
-1
```

ASC ASCII Value

Format	**ASC** (*string_expression*)
Group	String
Units	None
Data Type	String
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	CHR$
Product Revision	Command and Firmware Release

Description

This function returns the numeric code of the first character in the string expression.

Remarks

If the string is of zero length, the function returns zero.

See the following ASCII Table.

Examples

Example

```
PRINT ASC("X")
```

Example Output

```
88
```

ASCII Table

Following is an ASCII Table with decimal, hexadecimal, and character equivalents.

DEC	HEX	CHAR
0	00	NUL
1	01	SOH
2	02	STX
3	03	EXT
4	04	EOT
5	05	ENQ
6	06	ACK
7	07	BEL
8	08	BS
9	09	HT
10	0A	LF
11	0B	VT
12	0C	FF
13	0D	CR
14	0E	SO
15	0F	S1
16	10	DLE
17	11	XON
18	12	DC2
19	13	XOFF
20	14	DC4
21	15	NAK
22	16	SYN
23	17	ETB
24	18	CAN
25	19	EM
26	1A	SUB
27	1B	ESC
28	1C	FS
29	1D	GS
30	1E	RSt
31	1F	US

DEC	HEX	CHAR
32	20	SPACE
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	`
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	∅
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?

DEC	HEX	CHAR
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_

DEC	HEX	CHAR
96	60	'
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
100	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	DEL

DEC	HEX	CHAR
128	80	Ç
129	81	ü
130	82	é
131	83	â
132	84	ä
133	85	à
134	86	å
135	87	ç
136	88	ê
137	89	ë
138	8A	è
139	8B	ï
140	8C	î
141	8D	ì
142	8E	Ä
143	8F	Å
144	90	É
145	91	æ
146	92	Æ
147	93	ô
148	94	î
149	95	ò
150	96	û
151	97	ù
152	98	ÿ
153	99	Ö
154	9A	Ü
155	9B	¢
156	9C	£
157	9D	¥
158	9E	Pt
159	9F	ƒ

DEC	HEX	CHAR
160	A0	á
161	A1	í
162	A2	ó
163	A3	ú
164	A4	ñ
165	A5	Ñ
166	A6	ª
167	A7	º
168	A8	¿
169	A9	⌐
170	AA	¬
171	AB	$^1/_2$
172	AC	$^1/_4$
173	AD	¡
174	AE	«
175	AF	»
176	B0	░
177	B1	▓
178	B2	█
179	B3	¦
180	B4	┤
181	B5	╡
182	B6	╢
183	B7	╖
184	B8	╕
185	B9	╣
186	BA	║
187	BB	╗
188	BC	╝
189	BD	╜
190	BE	╛
191	BF	┐

DEC	HEX	CHAR
192	C0	└
193	C1	┴
194	C2	┬
195	C3	├
196	C4	–
197	C5	†
198	C6	╞
199	C7	╟
200	C8	╚
201	C9	╔
202	CA	╩
203	CB	╦
204	CC	╠
205	CD	=
206	CE	╬
207	CF	╧
208	D0	╨
209	D1	╤
210	D2	╥
211	D3	╙
212	D4	╘
213	D5	╒
214	D6	╓
215	D7	╫
216	D8	╪
217	D9	┘
218	DA	┌
219	DB	█
220	DC	▄
221	DD	▌
222	DE	▐
223	DF	▀

DEC	HEX	CHAR
224	E0	α
225	E1	β
226	E2	Γ
227	E3	π
228	E4	Σ
229	E5	σ
230	E6	∝
231	E7	τ
232	E8	Φ
233	E9	θ
234	EA	Ω
235	EB	δ
236	EC	∞
237	ED	∅
238	EE	∈
239	EF	∩
240	F0	≡
241	F1	±
242	F2	≥
243	F3	≤
244	F4	Ï
245	F5	|
246	F6	÷
247	F7	≈
248	F8	°
249	F9	•
250	FA	·
251	FB	√
252	FC	η
253	FD	2
254	FE	●
255	FF	

ASIN Arc Sine

Format	**ASIN** (*expression*)
Group	Trigonometric
Units	None
Data Type	FP32 or Long
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	SIN, COS, TAN, COT, ACOS, ATAN, ACOT
Product Revision	Command and Firmware Release

This function returns the arc sine of the expression.

ASINH Hyperbolic Arc Sine

Format	**ASINH** (*expression*)
Group	Hyperbolic
Units	None
Data Type	FP32 or Long
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	SINH, COSH, TANH, COTH, ACOSH, ATANH, ACOTH
Product Revision	Command and Firmware Release

This function returns the hyperbolic arc sine of the expression.

ATAN Arc Tangent

Format	**ATAN** (*expression*)
Group	Trigonometric
Units	None
Data Type	FP32 or Long
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	SIN, COS, TAN, COT, ASIN, ACOS, ACOT
Product Revision	Command and Firmware Release

This function returns the arc tangent of the expression.

ATANH Hyperbolic Arc Tangent

Format	**ATANH** (*expression*)
Group	Hyperbolic
Units	None
Data Type	FP32 or Long
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	SINH, COSH, TANH, COTH, ASINH, ACOSH, ACOTH
Product Revision	Command and Firmware Release

This function returns the hyperbolic arc tangent of the expression.

BIT Bit Flag Status

Format 1:	**BIT** *index*
Format 2:	**BIT** (*expression*)
Group	Logical
Units	None
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	AND, NAND, OR, NOR, XOR, XNOR, NOT
Product Revision	Command and Firmware Release

This function returns the state of a bit flag. The result is -1 if the bit is set and 0 if the bit is clear. This allows the result to be used in logical expressions.

Example

```
SET 128 : PRINT BIT 128
CLR 128 : PRINT BIT 128
```

Example Output

```
-1
0
```

CEIL Smallest Integer >= Expression

Format	**CEIL** (*expression*)
Group	Miscellaneous
Units	None
Data Type	Long
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	FLOOR, ROUND, TRUNC
Product Revision	Command and Firmware Release

This function returns the smallest integral value greater than or equal to the expression. The expression is rounded toward positive infinity.

Example

```
PRINT CEIL(1.25)
PRINT CEIL(1.75)
PRINT CEIL(-1.25)
PRINT CEIL(-1.75)
```

Example Output

```
2
2
-1
-1
```

CHR$ Character String

Format	**CHR$** (*code*)
Group	String
Units	None
Data Type	String
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	ASC
Product Revision	Command and Firmware Release

This function returns a string of one character as defined by the given "code". The valid range for "code" is 0 to 255. If the value is outside that range, an error is returned.

Example

```
PRINT CHR$(88)
```

Example Output

```
X
```

COS Cosine

Format	**COS** (*expression*)
Group	Trigonometric
Units	None
Data Type	FP32 or Long
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	SIN, TAN, COT, ASIN, ACOS, ATAN, ACOT
Product Revision	Command and Firmware Release

This function returns the cosine of the expression.

COSH Hyperbolic Cosine

Format	**COSH** (*expression*)
Group	Hyperbolic
Units	None
Data Type	FP32 or Long
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	SINH, TANH, COTH, ASINH, ACOSH, ATANH, ACOTH
Product Revision	Command and Firmware Release

This function returns the hyperbolic cosine of the expression.

COT Cotangent

Format	**COT** (*expression*)
Group	Trigonometric
Units	None
Data Type	FP32 or Long
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	SIN, COS, TAN, ASIN, ACOS, ATAN, ACOT
Product Revision	Command and Firmware Release

This function returns the cotangent of the expression.

COTH Hyperbolic Cotangent

Format	**COTH** (*expression*)
Group	Hyperbolic
Units	None
Data Type	FP32 or Long
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	SINH, COSH, TANH, ASINH, ACOSH, ATANH, ACOTH
Product Revision	Command and Firmware Release

This function returns the hyperbolic cotangent of the expression.

FLOOR Largest Integer <= Expression

Format	**FLOOR** (*expression*)
Group	Miscellaneous
Units	None
Data Type	Long
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	CEIL, ROUND, TRUNC
Product Revision	Command and Firmware Release

This function returns the largest integral value less than or equal to the expression. The expression is rounded toward negative infinity.

Example

```
PRINT FLOOR(1.25)
PRINT FLOOR(1.75)
PRINT FLOOR(-1.25)
PRINT FLOOR(-1.75)
```

Example Output

```
1
1
-2
-2
```

GETCH Wait For A Character

Format	**GETCH** (*device_number*)
Group	String
Units	None
Data Type	String
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	KBHIT, INKEY$, OPEN
Product Revision	Command and Firmware Release

This function returns a one character string from a device. If there is no character waiting to be read from the device, the function will wait until one becomes available.

The valid range for "device_number" is 0 to 3. Each program has its own device zero which is used as its default device. Devices one through three are controller-wide system resources that can be opened and used from within any program or from any system or program prompt.

Example

```
DIM $V(1,10)
OPEN "COM1:9600,N,8,1" AS #1
PRINT #1,
PRINT #1, "Press any key to continue"
$V0 = GETCH(1)
PRINT #1, "Program terminated"
CLOSE #1
```

INKEY$ Return A Character

Format	**INKEY$** (*device_number*)
Group	String
Units	None
Data Type	string
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	KBHIT, OPEN
Product Revision	Command and Firmware Release

Description

This function returns a one character string from a device. If there is no character waiting to be read from the device, the function will return a null string.

Remarks

The valid range for "device_number" is 0 to 3. Each program has its own device zero which is used as its default device. Devices one through three are controller-wide system resources that can be opened and used from within any program or from any system or program prompt.

Example

Example

```
DIM $V(1,10)
OPEN "COM1:9600,N,8,1" AS #1
_LOOP1
$V0 = UCASE$(INKEY$(1))
IF ($V0 = "A") THEN PRINT #1, "Apple"
IF ($V0 = "B") THEN PRINT #1, "Banana"
IF ($V0 = "C") THEN PRINT #1, "Coconut"
IF ($V0 = "X") THEN GOTO LOOP2
GOTO LOOP1
_LOOP2
PRINT #1, "Program terminated"
CLOSE #1
```

INSTR String Search

Format	**INSTR** (*string_expression1*, *string_expression2*)
Group	String
Units	None
Data Type	String or Long
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	N/A
Product Revision	Command and Firmware Release

This function returns the position of "string_expression2" within "string_expression1". If the second string can not be located within the first, the function returns zero.

If the first string is a null string, the function returns a zero. If the second string is a null string and the first string has a length greater than zero, the function returns a one.

Example

```
PRINT INSTR("ABCDEFG","CDE")
```

Example Output

```
3
```

KBHIT Check for Waiting Character

Format	**KBHIT** (*device_number*)
Group	String
Units	None
Data Type	Boolean return
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	INKEY$, OPEN
Product Revision	Command and Firmware Release

Description

This function checks a device to see if a character is waiting to be read. If there is no character waiting to be read, the function will return a zero. Otherwise, the function will return a negative one (-1) indicating success.

Remarks

The valid range for "device_number" is 0 to 3. Each program has its own device zero which is used as its default device. Devices one through three are controller-wide system resources that can be opened and used from within any program or from any system or program prompt.

Examples

Example 1

```
PROGRAM
REM --- main program
DIM $V(1,80)    : REM allocate 1 string variable of 80 characters
OPEN "COM1:9600,N,8,1" AS #1    : REM open COM port 1 as Device #1
REM at 9600 baud rate, N parity, 8 databits, 1 stopbit
PRINT #1, ""    : REM carriage return to next line

_Start
IF (KBHIT(1))    : REM check for waiting character (keyboard hit)
       $V0=INKEY$(1)    : REM read the character waiting in Device #1
       IF (ASC($V0)=27)    : REM check for ESC key
       REM if so, close port and end the program
       PRINT #1, "Closing port… Ending program. Bye."
       CLOSE #1    : REM close the Device #1 connection
       END
       ELSE
       PRINT #1,$V0;"=ASCII ";ASC($V0)    : REM print out the
       REM character received and its ASCII value
       ENDIF
       ENDIF
GOTO Start

ENDP
```

Example 2

Prints randomly-generated capital letters until a single character is sent. Program terminates when Exit is sent.

```
PROGRAM
'Program 0
REM main program
DIM #V (1,80)    : REM allocates 1 variable of 80 characters
OPEN "COM1:9600,N,8,1" AS #1    : REM open COM port 1 at
REM baud rate, N parity, 8 databits, and 1 stopbit
PRINT #1,""    : REM carriage return to next line
_LOOP1
PRINT #1, CHR$(65 + RND(26));
IF (KBHIT(1))  THEN GOTO LOOP2
DWL 1
GOTO LOOP1
_LOOP2
GOSUB LOOP3: REM fetch command
IF (UCASE$($V0) = "EXIT") THEN GOTO EXIT
GOTO LOOP1
_LOOP3
REM command input
PRINT #1, ""
INPUT #1, "Command?", $V0
RETURN
_Exit
REM program shutdown
PRINT #1, "Program terminated"
CLOSE #1

ENDP
```

LCASE$ Convert To Lower Case

Format	**LCASE$** (*string_expression*)
Group	String
Units	None
Data Type	String
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	UCASE$
Product Revision	Command and Firmware Release

This function returns a string with all letters in lower case. This function is useful for making string comparisons that are not case sensitive.

Example

```
PRINT LCASE$("AbCdEfG")
```

Example Output

```
abcdefg
```

LEFT$ Left String

Format	**LEFT$** (*string_expression*, *n*)
Group	String
Units	None
Data Type	n=Long; string_expression= string, string return
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	MID$, RIGHT$
Product Revision	Command and Firmware Release

This function returns the leftmost "n" characters of the given string. If "n" is greater than the length of the string, the entire string is returned.

Example

```
PRINT LEFT$("ABCDEFG", 3)
```

Example Output

```
ABC
```

LEN String Length

Format	**LEN** (*string_expression*)
Group	String
Units	None
Data Type	string_expression=string, long return
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	MID$, RIGHT$
Product Revision	Command and Firmware Release

This function returns the length of the given string expression.

Example

```
PRINT LEN("ABCDEFG")
```

Example Output

```
7
```

LN Natural Logarithm

Format	**LN** (*expression*)
Group	Miscellaneous
Units	None
Data Type	FP32 or Long
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	LOG
Product Revision	Command and Firmware Release

This function returns the natural logarithm of the expression.

LOG Common Logarithm

Format	**LOG** (*expression*)
Group	Miscellaneous
Units	None
Data Type	FP32 or Long
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	LN
Product Revision	Command and Firmware Release

This function returns the common logarithm of the expression.

MID$ Middle String

Format	**MID$** (*string_expression*, start, *length*)
Group	String
Units	None
Data Type	string_expression=string, start; length=Long, string return
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	LEFT$, RIGHT$
Product Revision	Command and Firmware Release

This function returns characters from the middle of the given string. If "start" is greater than the length of the string, the function returns a null string. If "length" would go beyond the end of the string, the function returns only the characters from "start" to the end of the string.

Example

```
PRINT MID$("ABCDEFG", 2, 5)
```

Example Output

```
BCDEF
```

MOD Modulus

Format	expression1 **MOD** expression2
Group	Arithmetic
Units	None
Data Type	FP32 or Long
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	*, /
Product Revision	Command and Firmware Release

This operator returns the modulus of the two expressions. The modulus is the remainder after dividing "expression1" by "expression2" an integral number of times. If the second expression evaluates to zero, the **MOD** operator returns 0.0. Otherwise, the modulus is calculated according to the following formula:

X **MOD** Y = X - **FLOOR** (X / Y) * Y

Example

```
PRINT 0.7 MOD 0.3
PRINT 0.7 MOD -0.3
PRINT -0.7 MOD 0.3
PRINT -0.7 MOD -0.3
```

Example Output

```
0.1
-0.2
0.2
-0.1
```

NAND Logical NAND

Format	expression1 **NAND** expression2
Group	Logical
Units	None
Data Type	Long
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	AND, OR, NOR, XOR, XNOR, NOT, BIT
Product Revision	Command and Firmware Release

This operator returns the logical **NAND** of the two expressions. Bits in the result will be set if the corresponding expression bits are not both set.

Example

```
PRINT 0 NAND 0
PRINT 0 NAND -1
PRINT -1 NAND 0
PRINT -1 NAND -1
```

Example Output

```
-1
-1
-1
0
```

NOR Logical NOR

Format	expression1 **NOR** expression2
Group	Logical
Units	None
Data Type	Long
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	AND, NAND, OR, NOR, XOR, XNOR, NOT, BIT
Product Revision	Command and Firmware Release

This operator returns the logical **NOR** of the two expressions. Bits in the result will be set if neither of the corresponding expression bits are set.

Example

```
PRINT 0 NOR 0
PRINT 0 NOR -1
PRINT -1 NOR 0
PRINT -1 NOR -1
```

Example Output

```
-1
0
0
0
```

NOT Bitwise Complement

Format	**NOT** *expression*
Group	Logical
Units	None
Data Type	Long
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	AND, NAND, OR, NOR, XOR, XNOR, BIT
Product Revision	Command and Firmware Release

This operator returns the logical **NOT** of the two expressions. Bits in the result will be set if the corresponding expression bits are clear.

Example

```
PRINT NOT 0
PRINT NOT -1
```

Example Output

```
-1
0
```

OR Logical OR

Format	expression1 **OR** expression2
Group	Logical
Units	None
Data Type	Long
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	AND, NAND, NOR, XOR, XNOR, NOT, BIT
Product Revision	Command and Firmware Release

This operator returns the logical **OR** of the two expressions. Bits in the result will be set if any of the corresponding expression bits are set.

Example

```
PRINT 0 OR 0
PRINT 0 OR -1
PRINT -1 OR 0
PRINT -1 OR -1
```

Example Output

```
0
1
1
1
```

RIGHT$ Right String

Format	**RIGHT$** (*string_expression*, *length*)
Group	String
Units	None
Data Type	string_expression=string; length=long, string return
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	LEFT$, MID$
Product Revision	Command and Firmware Release

This function returns the rightmost "n" characters of the given string. If "n" is greater than the length of the string, the entire string is returned.

Example

```
PRINT RIGHT$("ABCDEFG", 3)
```

Example Output

```
EFG
```

RND Random Integer

Format	**RND** (*expression*)
Group	Miscellaneous
Units	None
Data Type	Long
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	ROUND
Product Revision	Command and Firmware Release

This function returns a random integer between 0 and "expression" - 1.

Example

```
_LOOP1
PRINT RND(10); " ";
GOTO LOOP1
LRUN
```

Example Output

```
2 5 7 3 9 0 1 3 5 7 3 8 9 7 8 3 1 0 5 4 6 8 2 ...
```

ROUND Round To Nearest Integer

Format	**ROUND** (*expression*)
Group	Miscellaneous
Units	None
Data Type	Expression=FP32, Long return
Default	
Prompt Level	
See Also	CEIL, FLOOR, TRUNC
Product Revision	Command and Firmware Release

This function returns the nearest integral value to the expression.

Example

```
PRINT ROUND(1.25)
PRINT ROUND(1.75)
PRINT ROUND(-1.25)
PRINT ROUND(-1.75)
```

Example Output

```
1
2
-1
-2
```

SIN Sine

Format	**SIN** (*expression*)
Group	Trigonometric
Units	None
Data Type	FP32 or Long
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	COS, TAN, COT, ASIN, ACOS, ATAN, ACOT
Product Revision	Command and Firmware Release

This function returns the sine of the expression.

SINH Hyperbolic Sine

Format	**SINH** (*expression*)
Group	Hyperbolic
Units	None
Data Type	FP32 or Long
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	COSH, TANH, COTH, ASINH, ACOSH, ATANH, ACOTH
Product Revision	Command and Firmware Release

This function returns hyperbolic sine the of the expression.

SPACE$ String of Spaces

Format	**SPACE$** (*n*)
Group	String
Units	None
Data Type	String return, n=Long
Default	
Prompt Level	
See Also	STRING$
Product Revision	Command and Firmware Release

This function returns a string of "n" spaces.

Example 1

```
PRINT "**********"
PRINT "*"; SPACE$(8); "*"
PRINT "**********"
PRINT “0123456789”
PRINT “0”; SPACE$(8); “9”
PRINT “0123456789”
ENDP

P03>LRUN
**********
*        *
**********
0123456789
0        9
0123456789
P03>
```

SQRT Square Root

Format	**SQRT** (*expression*)
Group	Miscellaneous
Units	None
Data Type	FP32 or Long
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	+, –, *, /, **, MOD
Product Revision	Command and Firmware Release

This function returns the square root of the expression.

STR$ Convert Numeric To String

Format	**STR$** (*value*)
Group	String
Units	None
Data Type	Value=FP32, string return
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	STRING$, VAL
Product Revision	Command and Firmware Release

This function converts "value" to a string and returns the string.

Example

```
DIM $V(1, 10)
$V0 = STR$(1.234)
PRINT $V0
LRUN
```

Example Output

```
1.234
```

STRING$ String of Characters

Format 1:	**STRING$** (*length*, *code*)
Format 2:	**STRING$** (*length*, *string_expression*)
Group	String
Units	None
Data Type	Length=Long; code=0-2555 Long; string_expression= string, string return
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	SPACE$
Product Revision	Command and Firmware Release

This function returns a string of characters either defined by the given ASCII code or the first character of a string expression.

Example

```
PRINT STRING$(5, 88)
PRINT STRING$(10, "*")
LRUN
```

Example Output

```
XXXXX
**********
```

TAN Tangent

Format	**TAN** (*expression*)
Group	Trigonometric
Units	None
Data Type	FP32 or Long
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	SIN, COS, COT, ASIN, ACOS, ATAN, ACOT
Product Revision	Command and Firmware Release

This function returns the tangent of the expression.

TANH Hyperbolic Tangent

Format	**TANH** (*expression*)
Group	Hyperbolic
Units	None
Data Type	FP32 or Long
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	SINH, COSH, COTH, ASINH, ACOSH, ATANH, ACOTH
Product Revision	Command and Firmware Release

This function returns the hyperbolic tangent of the expression.

TRUNC Remove Fractional Part

Format	**TRUNC** (*expression*)
Group	Miscellaneous
Units	None
Data Type	Expression=FP32, long return
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	CEIL, FLOOR, ROUND
Product Revision	Command and Firmware Release

This function removes any fractional part of the expression and returns an integral result. The expression is rounded toward 0.0.

Example

```
PRINT TRUNC(1.25)
PRINT TRUNC(1.75)
PRINT TRUNC(-1.25)
PRINT TRUNC(-1.75)
```

Example Output

```
1
1
-1
-1
```

UCASE$ Convert To Upper Case

Format	**UCASE$** (*string_expression*)
Group	String
Units	None
Data Type	Expression=FP32, Long return
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	LCASE$
Product Revision	Command and Firmware Release

This function returns a string with all letters in upper case. This function is useful for making string comparisons that are not case sensitive.

Example

```
PRINT UCASE$("AbCdEfG")
```

Example Output

```
ABCDEFG
```

VAL Convert String To Numeric

Format	**VAL** (*string_expression*)
Group	String
Units	None
Data Type	string_expression=string, FP32 return
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	STR$
Product Revision	Command and Firmware Release

This function converts the "string_expression" to a numeric value and returns the value. Leading spaces and tab characters are ignored and the conversion continues until a character is reached that cannot be recognized as part of a number. If the conversion fails, the function returns a zero.

Example

```
DIM DV(1)
DV0 = VAL("1.234")
PRINT DV0
LRUN
```

Example Output

```
1.234
```

XNOR Logical XNOR

Format	expression1 **XNOR** expression2
Group	Logical
Units	None
Data Type	Long
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	AND, NAND, OR, NOR, XOR, NOT, BIT
Product Revision	Command and Firmware Release

This operator returns the logical **XNOR** of the two expressions. Bits in the result will be set if the corresponding expression bits are both set or both clear.

Example

```
PRINT 0 XNOR 0
PRINT 0 XNOR -1
PRINT -1 XNOR 0
PRINT -1 XNOR -1
```

Example Output

```
-1
0
0
-1
```

XOR Logical XOR

Format	expression1 **XOR** expression2
Group	Logical
Units	None
Data Type	Long
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	AND, NAND, OR, NOR, XNOR, NOT, BIT
Product Revision	Command and Firmware Release

This operator returns the logical **XOR** of the two expressions. Bits in the result will be set if only one of the corresponding expression bits is set.

Example

```
PRINT 0 XOR 0
PRINT 0 XOR -1
PRINT -1 XOR 0
PRINT -1 XOR -1
```

Example Output

```
0
-1
-1
0
```

PLC Reference

The PLC reference helps you create PLC programs for the ACR series controller.

- PLC Programming
- PLC Commands
- PLC Instructions

(PLC commands are not valid for the Aries Controller.)

PLC Programming

PLC programs are created in the same manner as user programs, but with a limited instruction set that is compiled into machine code for high-speed execution. Each PLC program can contain a maximum of 100 (200 for ACR8010, ACR1505, and ACR9000) instructions. Memory for the PLC programs must be dimensioned from the system level using the **DIM PLC** command. On average, dimensioning 32 bytes per PLC instruction is sufficient.

PLC programs are linked into the PLC scanner, which is a list of events that are to be executed at the servo interrupt rate. During each servo interrupt (**PERIOD**), a single event from this list is executed. During the next interrupt, the next event is executed. This process is repeated after the last item in the list. In addition to PLC programs, the scanner event list also contains an input/output/clock scan and a timer/counter/latch scan.

As an example, two PLC programs running at the default 500 microsecond servo rate will be serviced every 2 milliseconds. One interrupt for the input/output/clock scan, one interrupt for the timer/counter/latch scan, and one interrupt for each of the PLC program scans. All eight PLC programs would be scanned every 5 milliseconds.

Related Parameters

Individual PLC programs may also be instructed to scan at a rate slower than the servo interrupt rate by setting system parameters. The "tick preload" parameter controls the scan rate in milliseconds and the "tick count" indicates the number of milliseconds remaining before the PLC program is scanned.

The following table outlines parameters related to PLC operation.

PLC Number	Tick Preload	Tick Count
0	P6656	P6657
1	P6672	P6673
2	P6688	P6689
3	P6704	P6705
4	P6720	P6721
5	P6736	P6737
6	P6752	P6753
7	P6768	P6769

Related Flags

The "PLC Running" flag is set when the **RUN** command is executed and cleared when the **HALT** command is executed. The "First PLC Scan" flag is set when a PLC program is **RUN** and cleared after the first PLC scan. The contact of this relay can be used for PLC reset logic as needed. The "RUN Request" and "HALT Request" flags cause the execution of **RUN** and **HALT** commands respectively.

The following table outlines bit flags related to PLC operation.

PLC Number	PLC Running	First PLC Scan	Run Request	Halt Request
0	Bit1536	Bit1537	Bit1538	Bit1539
1	Bit1568	Bit1569	Bit1570	Bit1571
2	Bit1600	Bit1601	Bit1602	Bit1603
3	Bit1632	Bit1633	Bit1634	Bit1635
4	Bit1664	Bit1665	Bit1666	Bit1667
5	Bit1696	Bit1697	Bit1698	Bit1699
6	Bit1728	Bit1729	Bit1730	Bit1731
7	Bit1760	Bit1761	Bit1762	Bit1763

PLC Commands

PLC commands control the operation of PLC programs. (PLC commands are not valid for the Aries Controller.)

- The **PLC**, **PON**, and **POFF** commands can be executed from any prompt.
- The **RUN**, **HALT**, **LIST**, and **MEM** commands are similar to their user program counterparts, but they act slightly different when executed from the **PLC** prompt.

The following is a list of commands related to PLC programming:

Command	Description
HALT	Halt an executing program
LIST	List a stored program
MEM	Display memory allocation
PLC	Switch to a PLC prompt
PON	Turn on PLC scanning
POFF	Turn off PLC scanning
RUN	Run a stored program

HALT Halt PLC Program

Format	**HALT**
Group	Program Control
Units	None
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	RUN
Product Revision	Command and Firmware Release

This command removes the current PLC program from the PLC scanner. If the PLC scanner is idling as the result of a **POFF** command, the current PLC program will not be put back into the scanner list when the **PON** command is executed. Halting a PLC program does not cause other PLC programs or the timer/counter/latch update event to be removed from the PLC scanner list.

Example

```
PLC1>HALT
```

LIST List PLC Program

Format	**LIST** {*first*} {, {*last*}}
Group	Program Control
Units	None
Data Type	Long
Default	N/A
Prompt Level	PROGx, PLCx
See Also	RUN, HALT
Product Revision	Command and Firmware Release

This command lists the currently selected PLC program. The listings of **CNT** and **TIM** preload values reflect the current setting of the corresponding system parameters. The *first* and *last* arguments define listing ranges as follows:

LIST *first*	Lists a single line.
LIST *first, last*	Lists from "first" to "last".
LIST *first,*	Lists from "first" to end of program.
LIST *,last*	Lists form start of program to "last".
LIST	Lists entire program.

Example

```
LIST 100,199
```

MEM Show PLC Memory

Format	**MEM**
Group	Memory Control
Units	Bytes
Data Type	Long
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	CLEAR, DIM
Product Revision	Command and Firmware Release

This command displays the amount of free memory remaining in the current PLC space. Running a PLC requires a certain amount of free memory to store the machine code generated during compilation. On average, a total of 32 bytes of storage are required for each PLC instruction, 8 for the source and 24 for the machine code. The memory that is displayed by this command only reflects the memory used by the source storage, even if the PLC has been compiled and is currently running.

Example

```
MEM
```

PLC Switch To PLC Program

Format	**PLC** {*number*}
Group	Operating System
Units	None
Data Type	N/A
Default	N/A
Prompt Level	SYS, PROGx, PLCx
See Also	DIM, PROG, SYS
Product Revision	Command and Firmware Release

This command switches the communications channel to the designated **PLC** prompt. The *number* argument indicates which PLC and is in the range of 0 to 7. The command prompt keeps track of the current level as follows:

```
SYS>PROG3

P03>PLC5

PLC5>SYS

SYS>_
```

The system must be at the **PLC** prompt in order to run and edit PLC programs. The memory for the PLC must have been dimensioned from the system level with the **DIM PLC** command before PLC programs can be entered.

Example

```
PLC 2
```

PON Turn On PLC Scanning

Format	**PON**
Group	PLC commands
Units	None
Data Type	N/A
Default	OFF
Prompt Level	PLCx
See Also	RUN, HALT
Product Revision	Command and Firmware Release

This command initializes the PLC scanner list to include the input/output/clock update event, any compiled PLC programs which may have been set to an idle state with the **POFF** command, and the timer/counter/latch update event. Running a PLC program will also cause this initialization to take place.

The input/output/clock update event is always in the PLC scanner list even if a **POFF** command has been issued. As the name implies, this event updates the opto-isolated digital I/O, the global system clock (P6916), and the clock tick flags (Bit80-83).

Note that the **PON** command must also be executed if the bit flags and parameters for timers, counters, or latches are to be used from normal user programs. Otherwise, the objects will not be updated by the control.

Example

```
PON
```

POFF Turn Off PLC Scanning

Format	**POFF**
Group	PLC commands
Units	None
Data Type	N/A
Default	OFF
Prompt Level	PLCx
See Also	RUN, HALT
Product Revision	Command and Firmware Release

This command resets the PLC scanner list to contain only the input/output/clock update event. Currently running PLC programs are put in an idle state and will be put back into the PLC scanner list when a **PON** command is executed.

Example

```
POFF
```

RUN Run PLC Program

Format	**RUN**
Group	PLC commands
Units	None
Data Type	N/A
Default	OFF
Prompt Level	PLCx
See Also	HALT
Product Revision	Command and Firmware Release

This command will run the current PLC program and return to the command prompt. When a PLC program is run, the control first 'compiles' the PLC program source and then 'links' the result into the PLC scanner.

NOTE: PLC programs are limited to 100 instructions, excepting ACR1505, ACR8010, ACR8020, and ACR9000 which are limited to 200 instructions.

Compilation takes the PLC program source and generates high-speed machine code specific to the control's processor. This step requires a certain amount of memory to store the generated code. An error will be generated if there is not enough free memory available to compile the PLC program.

After the PLC program is successfully compiled, the program is linked to the PLC scanner. The 'scanner' is a list of high-speed events. During each servo interrupt, a single event from this list is executed. During the next interrupt, the next event is executed. This process is repeated after the last item in the list.

Note that running a PLC program also executes an implied **PON** command, activating the timer/counter/latch update event and any pending PLC programs that may have been set to an idle state with the **POFF** command.

Example

```
RUN
```

PLC Instructions

PLC instructions are combined to create PLC programs. Each instruction represents either the contact or coil of a relay on a ladder logic diagram. In the description of these instructions, a relay is a bit flag, a contact is an instruction that monitors the state of a bit flag, and a coil is an instruction that controls the state of a bit flag.

For more information about command formats, arguments, and syntax, see Command Syntax.

The following is a list of instructions related to PLC programming:

Command	Description
AND	Add NO contact in series
AND LD	Connect blocks in series
AND NOT	Add NC contact in series
CNT	Connect blocks to counter
CTD	Connect blocks to counter
END	End of PLC ladder
KR	Connect blocks to latch
LAT	Connect blocks to latch
LD	Start block with NO contact
LD NOT	Start block with NC contact
OR	Add NO contact in parallel
OR LD	Connect blocks in parallel
OR NOT	Add NC contact in parallel
OUT	Connect block to coil
PBOOT	Activate PLC on power up
TIM	Connect block to timer
TM	Connect block to timer

AND Add NO Contact In Series

Formats	**AND** *contact* **AND TIM** *timer* **AND CNT** counter **AND KR** *latch*
Group	PLC Instructions
Units	None
Data Type	Boolean or Long (Timers and Counters only)
Default	N/A
Prompt Level	PLCx
See Also	AND LD, AND NOT
Related Topics	N/A
Product Revision	Command and Firmware Release

This instruction connects a normally open contact in series with the current logic block. An error will be generated if there are no logic blocks open at that point in the PLC program. The ***contact*** argument can be any bit flag. The other formats indicate the output contacts of global PLC timers, counters, and latches.

Example

```
10 LD 00
20 AND 01
30 OUT 32
```

AND LD Connect Blocks In Series

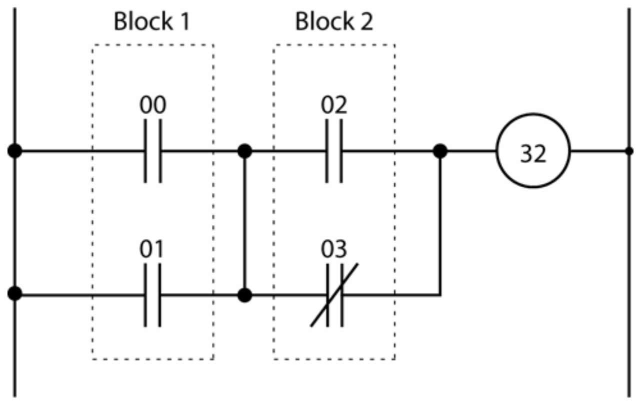
Format	**AND LD**
Group	PLC Instructions
Units	None
Data Type	N/A
Default	N/A
Prompt Level	PLCx
See Also	AND, AND NOT
Related Topics	N/A
Product Revision	Command and Firmware Release

This instruction takes the two most recent logic blocks and connects them in series, creating a new logic block. An error will be generated if there are not at least two logic blocks open at that point in the PLC program.

Example Logic

In this example, two normally open contacts from relays 00 and 01 are connected in parallel to form a block. Then a normally open contact from relay 02 and a normally closed contact from relay 03 are connected in parallel to form a second block. The two blocks are then combined in series and connected to the coil of relay 32.

The following illustrates AND LD logic:



Example

The following PLC code fragment implements the ladder logic shown above. Lines 100 and 110 create the first logic block. Lines 120 and 130 create the second logic block. Line 140 combines the blocks in series. Line 150 connects the block to relay 32.

```
100 LD 00
110 OR 01
120 LD 02
130 OR NOT 03
140 AND LD
150 OUT 32
```

AND NOT Add NC Contact In Series

Formats	**AND NOT** *contact* **AND NOT TIM** *timer* **AND NOT CNT** *counter* **AND NOT KR** *latch*
Group	PLC Instructions
Units	None
Data Type	Boolean or Long (Timers and Counters only)
Default	N/A
Prompt Level	PLCx
See Also	AND, AND LD
Related Topics	N/A
Product Revision	Command and Firmware Release

This instruction connects a normally closed contact in series with the current logic block. An error will be generated if there are no logic blocks open at that point in the PLC program. The ***contact*** argument can be any bit flag. The other formats indicate the output contacts of global PLC timers, counters, and latches.

Example

```
10 LD 00
20 AND NOT 01
30 OUT 32
```

CNT Connect Blocks To Counter

Format	**CNT** *counter* {*preload*}
Group	PLC Instructions
Units	None
Data Type	Long
Default	0
Prompt Level	PLCx
See Also	CTD, LD, TM
Related Topics	PLC Flags
Product Revision	Command and Firmware Release

NOTE: This command is deprecated. Use the **CTD** command.

This instruction takes two logic blocks and connects them to the given 'counter'. The first block is connected to the 'clock' coil and the second block is connected to the 'reset' coil. There are eight global PLC counters. An error will be generated if there are not exactly two logic blocks open at that point in the PLC program.

The optional *preload* argument sets the counter preload parameter when the instruction is stored in the PLC. If the preload is not specified, the system parameter remains unchanged. When the PLC is listed, the **CNT** instruction will reflect the current counter preload setting if it has been changed by a direct parameter setting.

A counter decrements once on every rising edge of its clock input until it reaches zero. The counter produces an output when the count is zero. When the reset input of a counter is turned on, the counter is reset to its preload value and the output turns off. Clock inputs are ignored while the reset input is on. Both the current count and preload are retained in battery backup memory during power down.

When a PLC program is run, the program is scanned to make sure that the individual counters are not being controlled by multiple **CNT** instructions. Duplication checks are only done within a PLC program, not across multiple PLC boundaries. Counter output contacts can be used any number of times.

Related Information

The following table outlines parameters and bit flags related to PLC counters. These can be used by normal programs to control and monitor counters with or without any PLC programs running. Note that if counters are to be used without PLC programs, the **PON** command must still be executed to enable updating of the counters.

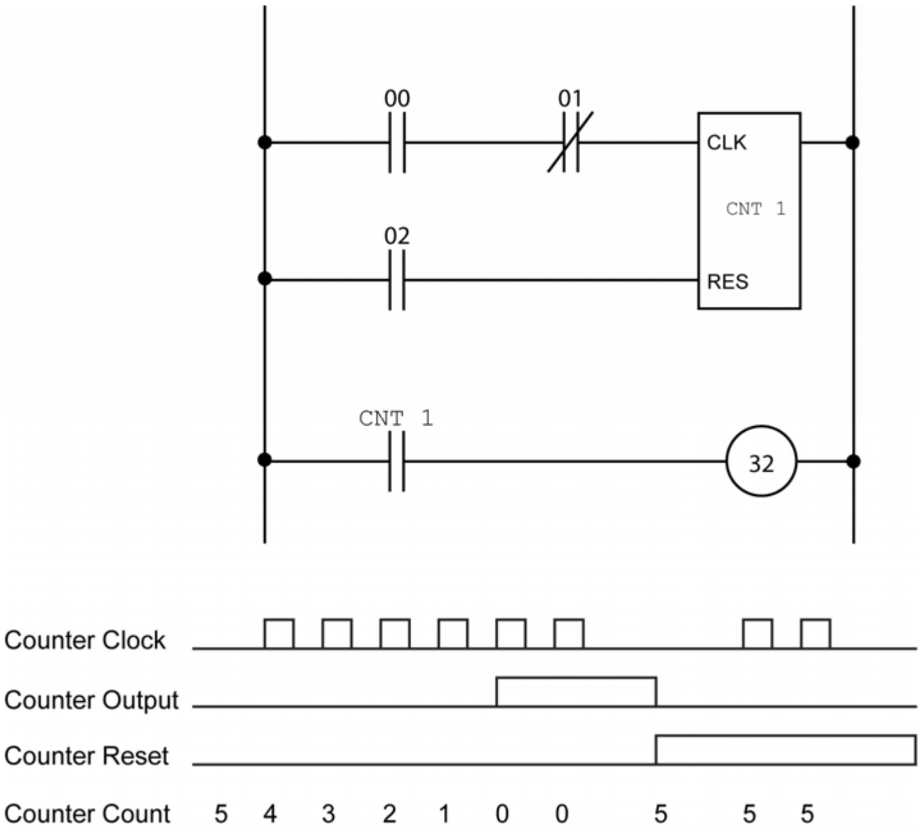
The following table provides a cross reference for the PLC counter:

Counter	Preload	Count	Output	Clock	Reset
0	P6662	P6663	Bit1556	Bit1557	Bit1558
1	P6678	P6679	Bit1588	Bit1589	Bit1590
2	P6694	P6695	Bit1620	Bit1621	Bit1622
3	P6710	P6711	Bit1652	Bit1653	Bit1654
4	P6726	P6727	Bit1684	Bit1685	Bit1686
5	P6742	P6743	Bit1716	Bit1717	Bit1718
6	P6758	P6759	Bit1748	Bit1749	Bit1750
7	P6774	P6775	Bit1780	Bit1781	Bit1782

Example Logic

In this example, a normally open contact from relay 00 and a normally closed contact from relay 01 are connected in series to form a block. Then a normally open contact from relay 02 forms a second block. These blocks are then connected to the clock and reset coils of counter 1 which is set to 5 counts. To bring out the state of the counter, a normally open contact from the counter is connected to the coil of relay 32.

The following illustrates a PLC counter:



Example

The following PLC code fragment implements the ladder logic shown above. Lines 100 and 110 create the first block. Line 120 creates the second block. Line 130 connects the two blocks to counter 1 and sets the count to 5. Lines 140 and 150 connect the counter output to relay 32.

```
100 LD 00
110 AND NOT 01
120 LD 02
130 CNT 1 5
140 LD CNT 1
150 OUT 32
```

CTD Connect blocks to counter

Format	**CTD** *counter reset* {*preload*}
Group	PLC Instructions
Units	None
Data Type	Long
Default	0
Prompt Level	PLCx
See Also	LD, TM
Related Topics	PLC Flags
Product Revision	Command and Firmware Release

The **CTD** command provides a more compact PLC program than the **CNT** command. It can do the following:

- Include the reset on the same line as the **CTD** command.
- Use multiple **CTD** function blocks on a single rung.

NOTE: To close a rung, you must use the **OUT** command.

The optional *preload* argument sets the counter preload parameter when the instruction is stored in the PLC. If the preload is not specified, the system parameter remains unchanged. When the PLC is listed, the **CTD** instruction will reflect the current counter preload setting if it has been changed by a direct parameter setting.

A counter decrements once on every rising edge of its clock input until it reaches zero. The counter produces an output when the count is zero. When the reset input of a counter is turned on, the counter is reset to its preload value and the output turns off. Clock inputs are ignored while the reset input is on. Both the current count and preload are retained in battery backup memory during power down.

When a PLC program is run, the program is scanned to make sure that the individual counters are not being controlled by multiple **CTD** instructions. Duplication checks are only done within a PLC program, not across multiple PLC boundaries. Counter output contacts can be used any number of times.

Related Information

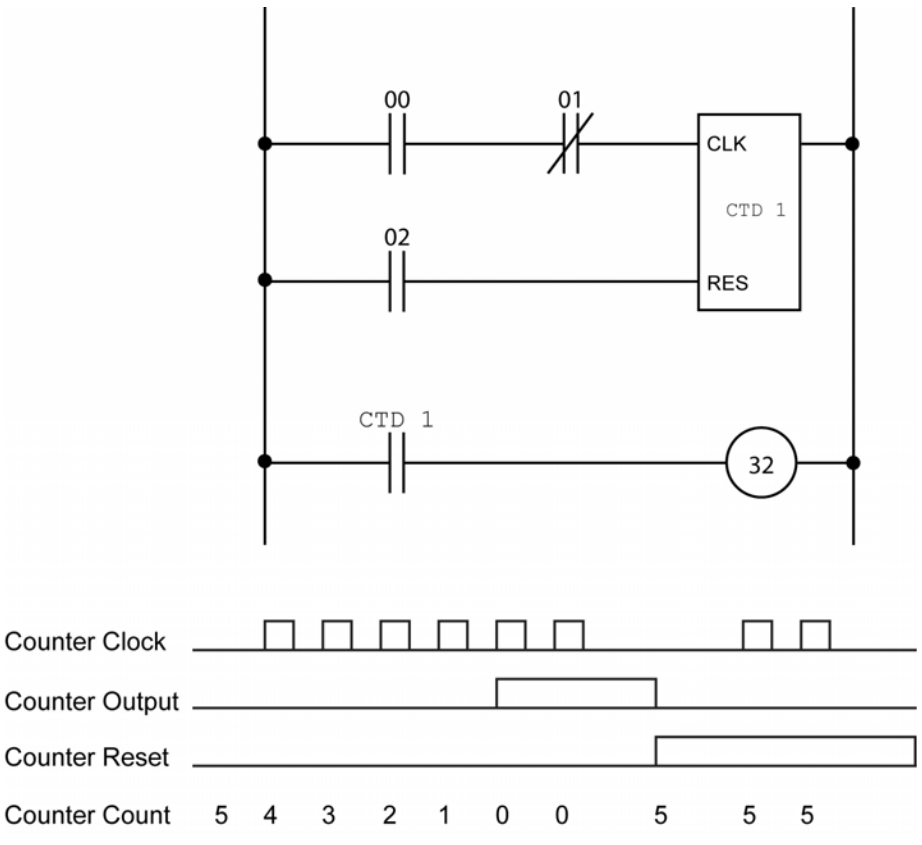
The following table outlines parameters and bit flags related to PLC counters. These can be used by normal programs to control and monitor counters with or without any PLC programs running. Note that if counters are to be used without PLC programs, the **PON** command must still be executed to enable updating of the counters.

The following table provides a cross reference for the PLC counter:

Counter	Preload	Count	Output	Clock	Reset
0	P6662	P6663	Bit1556	Bit1557	Bit1558
1	P6678	P6679	Bit1588	Bit1589	Bit1590
2	P6694	P6695	Bit1620	Bit1621	Bit1622
3	P6710	P6711	Bit1652	Bit1653	Bit1654
4	P6726	P6727	Bit1684	Bit1685	Bit1686
5	P6742	P6743	Bit1716	Bit1717	Bit1718
6	P6758	P6759	Bit1748	Bit1749	Bit1750
7	P6774	P6775	Bit1780	Bit1781	Bit1782

Example 1 (CNT command)

In this example, a normally open contact from relay 00 and a normally closed contact from relay 01 are connected in series to form a block. Then a normally open contact from relay 02 forms a second block. These blocks are then connected to the clock and reset coils of counter 1 which is set to 5 counts. To bring out the state of the counter, a normally open contact from the counter is connected to the coil of relay 32.

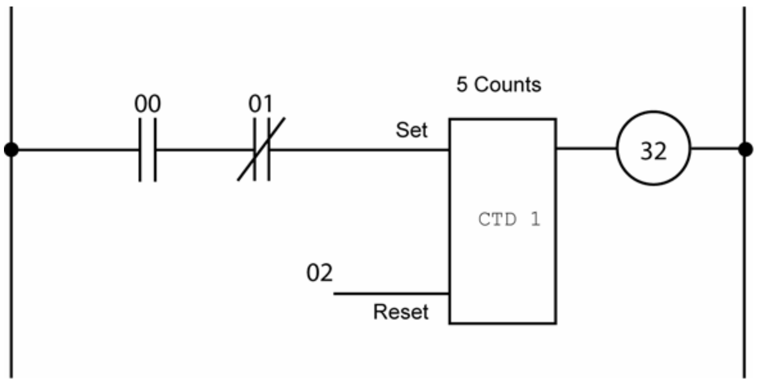


The following PLC code fragment implements the ladder logic as shown in the illustration above. Lines 100 and 110 create the first block. Line 120 creates the second block. Line 130 connects the two blocks to counter 1 and sets the count to 5. Lines 140 and 150 connect the counter output to relay 32.

```
100 LD 00
110 AND NOT 01
120 LD 02
130 CNT 1 5
140 LD CNT 1
150 OUT 32
```

Example 2 (CTD command)

The following illustrates a PLC program:



The following PLC code fragment implements the ladder logic as shown in the illustration above. Lines 100 and 110 create the first block. Line 130 connects the two blocks to counter 1, set the reset input to 2, and sets the count to 5. Line150 connects the counter output to relay 32.

```
100 LD 00
110 AND NOT 01
130 CTD 1 02 5
150 OUT 32
```

Example 3

Building on example 2, the following adds a second rung and another reset input in line 140.

```
100 LD 00
110 AND NOT 01
130 CTD 1 02 5
140 CTD 2 03 5
150 OUT 32
```

END End Of PLC Ladder

Format	**END**
Group	PLC Instructions
Units	None
Data Type	N/A
Default	N/A
Prompt Level	PLCx
See Also	RUN
Related Topics	N/A
Product Revision	Command and Firmware Release

This instruction indicates the end of the current PLC ladder. Although typically the last instruction in a PLC program, it may also be inserted in the middle to "cut off" the rest of the ladder. If there is no **END** instruction in a PLC program, the PLC will execute up to and including the last instruction.

Example

```
10 LD 00
20 OUT 32
30 END
40 LD 01
50 OUT 33
```

KR Connect Blocks To Latch

Format	**KR** *latch*
Group	PLC Instructions
Units	None
Data Type	N/A
Default	N/A
Prompt Level	PLCx
See Also	CTD, LAT, TM
Related Topics	PLC Flags
Product Revision	Command and Firmware Release

NOTE: This command is deprecated. Use the **LAT** command.

This instruction takes two logic blocks and connects them to the given 'latch'. The first block is connected to the 'set' coil and the second block is connected to the 'reset' coil. There are eight global PLC latches. An error will be generated if there are not exactly two logic blocks open at that point in the PLC program.

A latch output turns on when its set input is turned on. The latch output will remain on even after the set input goes away. When the reset input of a latch is turned on, the latch output will turn off. The set input is ignored when the reset input is turned on. The output state of a latch is retained in battery backup memory during power down.

When a PLC program is run, the program is scanned to make sure that the individual latches are not being controlled by multiple **KR** instructions. Duplication checks are only done within a PLC program, not across multiple PLC boundaries. Latch output contacts can be used any number of times.

Related Information

The following table outlines the bit flags related to PLC latches. These can be used by normal programs to control and monitor PLC latches with or without any PLC programs running. Note that if latches are to be used without PLC programs, the **PON** command must still be executed to enable updating of the latches.

The following table provides a cross reference for the PLC latch:

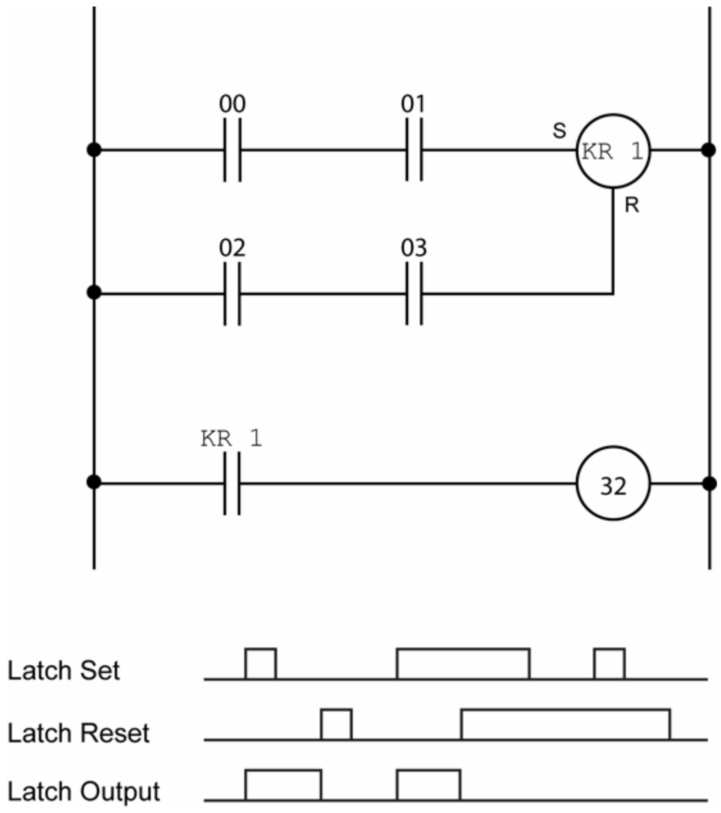
Latch	Output	Set	Reset
0	Bit1564	Bit1565	Bit1566
1	Bit1596	Bit1597	Bit1598
2	Bit1628	Bit1629	Bit1630
3	Bit1660	Bit1661	Bit1662
4	Bit1692	Bit1693	Bit1694
5	Bit1724	Bit1725	Bit1726
6	Bit1756	Bit1757	Bit1758

Latch	Output	Set	Reset
7	Bit1788	Bit1789	Bit1790

Example Logic

In this example, two normally open contacts from relays 00 and 01 are connected in series to form a block. Then two normally open contacts from relays 02 and 03 are connected in series to form a second block. These blocks are then connected to the set and reset coils of latch 1. To bring out the state of the latch, a normally open contact from the latch is connected to the coil of relay 32.

The following illustrates a PLC latch:



Example

The following PLC code fragment implements the ladder logic shown above. Lines 100 and 110 create the first block. Lines 120 and 130 create the second block. Line 140 connects the blocks to latch 1. Lines 150 and 160 connect the latch output to relay 32.

```
100 LD 00
110 AND 01
120 LD 02
130 AND 03
140 KR 1
150 LD KR 1
160 OUT 32
```

LAT Connect Blocks to Latch

Format	**LAT** *n reset*
Group	PLC Instructions
Units	None
Data Type	N/A
Default	N/A
Prompt Level	PLCx
See Also	CTD, TM
Related Topics	N/A
Product Revision	Command and Firmware Release

The **LAT** command provides a more compact PLC program than the **KR** command. It can do the following:

- Include the reset on the same line as the **CTD** command.
- Use multiple **LAT** function blocks on a single rung.

NOTE: To close a rung, you must use the **OUT** command.

A latch output turns on when its set input is turned on. The latch output will remain on even after the set input goes away. When the reset input of a latch is turned on, the latch output will turn off. The set input is ignored when the reset input is turned on. The output state of a latch is retained in battery backup memory during power down.

LD Start Block With NO Contact

Formats	**LD** *contact* **LD TIM** *timer* **LD CNT** *counter* **LD KR** *latch*
Group	PLC Instructions
Units	None
Data Type	Boolean or Long (Timers and Counters only)
Default	N/A
Prompt Level	PLCx
See Also	LD NOT
Related Topics	PLC Flags
Product Revision	Command and Firmware Release

This instruction opens a new logic block using a normally open contact. The ***contact*** argument can be any bit flag. The other formats indicate the output contacts of global PLC timers, counters, and latches.

Example

```
10 LD 00
20 OUT 32
```

LD NOT Start Block With NC Contact

Formats	**LD NOT** *contact*
	LD NOT TIM *timer*
	LD NOT CNT *counter*
	LD NOT KR *latch*
Group	PLC Instructions
Units	None
Data Type	Boolean or Long (Timers and Counters only)
Default	N/A
Prompt Level	PLCx
See Also	LD
Related Topics	N/A
Product Revision	Command and Firmware Release

This instruction opens a new logic block using a normally closed contact. The ***contact*** argument can be any bit flag. The other formats indicate the output contacts of global PLC timers, counters, and latches.

Example

```
10 LD NOT 00
20 OUT 32
```

OR Add NO Contact In Parallel

Formats	**OR** *contact* **OR TIM** *timer* **OR CNT** *counter* **OR KR** *latch*
Group	PLC Instructions
Units	None
Data Type	Boolean or Long (Timers and Counters only)
Default	N/A
Prompt Level	PLCx
See Also	AND, OR LD, OR NOT
Related Topics	N/A
Product Revision	Command and Firmware Release

This instruction connects a normally open contact in parallel across the current logic block. An error will be generated if there are no logic blocks open at that point in the PLC program. The ***contact*** argument can be any bit flag. The other formats indicate the output contacts of global PLC timers, counters, and latches.

Example

```
10 LD 00
20 OR 01
30 OUT 32
```

OR LD Connect Blocks In Parallel

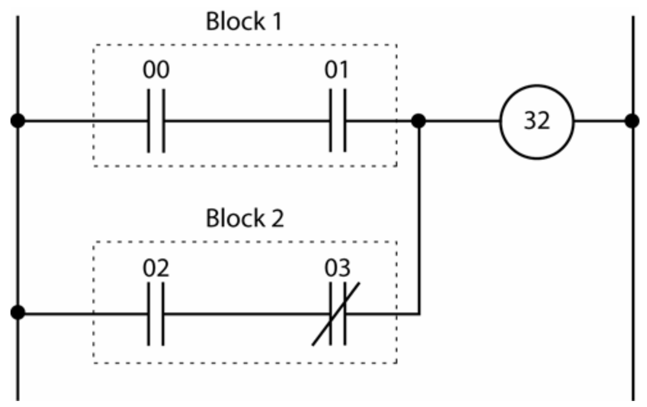
Format	**OR LD**
Group	PLC Instructions
Units	None
Data Type	N/A
Default	N/A
Prompt Level	PLCx
See Also	OR, OR NOT
Related Topics	N/A
Product Revision	Command and Firmware Release

This instruction takes the two most recent logic blocks and connects them in parallel, creating a new logic block. An error will be generated if there are not at least two logic blocks open at that point in the PLC program.

Example Logic

In this example, two normally open contacts from relays 00 and 01 are connected in series to form a block. Then a normally open contact from relay 02 and a normally closed contact from relay 03 are connected in series to form a second block. The two blocks are then combined in parallel and connected to the coil of relay 32.

The following illustrates OR LD logic:



Example

The following PLC code fragment implements the ladder logic shown above. Lines 100 and 110 create the first logic block. Lines 120 and 130 create the second logic block. Line 140 combines the blocks in parallel. Line 150 connects the block to relay 32.

```
100 LD 00
110 AND 01
120 LD 02
130 AND NOT 03
140 OR LD
```

OR NOT Add NC Contact In Parallel

Formats	**OR NOT** *contact* **OR NOT TIM** *timer* **OR NOT CNT** *counter* **OR NOT KR** *latch*
Group	PLC Instructions
Units	None
Data Type	Boolean or Long (Timers and Counters only)
Default	N/A
Prompt Level	PLCx
See Also	AND, OR LD
Related Topics	N/A
Product Revision	Command and Firmware Release

This instruction connects a normally closed contact in parallel across the current logic block. An error will be generated if there are no logic blocks open at that point in the PLC program. The ***contact*** argument can be any bit flag. The other formats indicate the output contacts of global PLC timers, counters, and latches.

Example

```
10 LD 00
20 OR NOT 01
30 OUT 32
150 OUT 32
```

OUT Connect Block To Coil

Format	**OUT** *coil*
Group	PLC Instructions
Units	None
Data Type	N/A
Default	N/A
Prompt Level	PLCx
See Also	LD
Related Topics	N/A
Product Revision	Command and Firmware Release

This instruction connects the current logic block to the coil of a relay (bit flag) and closes the logic block. The *coil* argument can be any bit flag index. An error will be generated if there is not exactly one logic block open at that point in the PLC program. Note that this error will not be generated in the case of multiple OUT instructions even though the current block is closed after the first OUT instruction.

When a PLC program is run, the program is scanned to make sure that individual relay coils are not being controlled by multiple OUT instructions. Duplication checks are only done within a PLC program, not across multiple PLC boundaries. Relay contacts can be used any number of times.

Example

```
10 LD 00
20 OUT 32
```

PBOOT Activate PLC On Power Up

Format	**PBOOT**
Group	PLC Instructions
Units	None
Data Type	N/A
Default	N/A
Prompt Level	PLCx
See Also	RUN
Related Topics	N/A
Product Revision	Command and Firmware Release

If used, this instruction should be the first instruction of a PLC program. On power up, the system checks the beginning of all PLC programs for a **PBOOT** instruction. If it finds one, the run request flag for that PLC is set.

Example

```
10 PBOOT
20 LD 00
0 OUT 32
```

TIM Connect Block To Timer

Format	**TIM** *timer* {*preload*}
Group	PLC Instructions
Units	Milliseconds
Data Type	Long
Default	0
Prompt Level	PLCx
See Also	CTD, LAT
Related Topics	PLC Flags
Product Revision	Command and Firmware Release

NOTE: This command is deprecated. Use the **TM** command.

This instruction connects the current logic block to the given 'timer' coil. There are eight global PLC timers. An error will be generated if there is not exactly one logic block open at that point in the PLC program.

The optional *preload* argument sets the timer preload parameter when the instruction is stored in the PLC. If the preload is not specified, the system parameter remains unchanged. When the PLC is listed, the **TIM** instruction will reflect the current timer preload setting if it has been changed by a direct parameter setting.

When a timer input is turned on, the timer count decrements once every millisecond until it reaches zero. The timer produces an output when the count is zero. When the input of a timer is turned off, the count is reset to its preload value and the output turns off. Timer counts and preloads are in milliseconds. The timer preload is retained in battery backup memory during power down, but the current timer count is not.

When a PLC program is run, the program is scanned to make sure that the individual timers are not being controlled by multiple **TIM** instructions. Duplication checks are only done within a PLC program, not across multiple PLC boundaries. Timer output contacts can be used any number of times.

Related Information

The following table outlines parameters and bit flags related to PLC timers. These can be used by normal programs to control and monitor PLC timers with or without any PLC programs running. Note that if timers are to be used without PLC programs, the **PON** command must still be executed to enable updating of the timers.

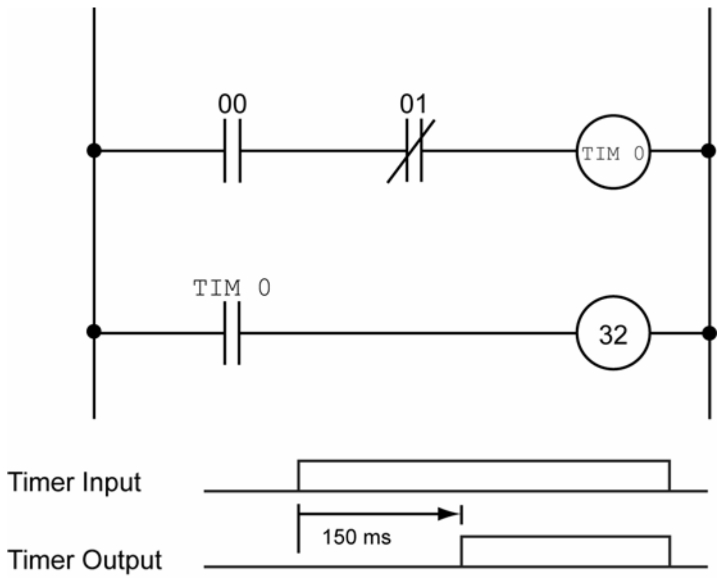
The following table provides a cross reference for the PLC timer:

Timer	Preload	Count	Output	Input
0	P6660	P6661	Bit1552	Bit1553
1	P6676	P6677	Bit1584	Bit1585
2	P6692	P6693	Bit1616	Bit1617
3	P6708	P6709	Bit1648	Bit1649
4	P6724	P6725	Bit1680	Bit1681
5	P6740	P6741	Bit1712	Bit1713
6	P6756	P6757	Bit1744	Bit1745
7	P6772	P6773	Bit1776	Bit1777

Example Logic

In this example, a normally open contact from relay 00 and a normally closed contact from relay 01 are connected in series to form a block. This block is then connected to the input coil of timer 0 which is set for 150 milliseconds. To bring out the state of the timer, a normally open contact from the timer is connected to the coil of relay 32.

The following illustrates a PLC timer:



Example

The following PLC code fragment implements the ladder logic shown above. Lines 100 and 110 create a new logic block. Line 120 connects the block to timer 0 and sets the timer to 150 milliseconds. Lines 130 and 140 connect the timer output to relay 32.

```
100 LD 00
110 AND NOT 01
120 TIM 0 150
130 LD TIM 0
140 OUT 32
```

TM Connect Block to Timer

Format	**TM** *timer* {*preload*}
Group	PLC Instructions
Units	Milliseconds
Data Type	Long
Default	0
Prompt Level	PLCx
See Also	TIM
Related Topics	PLC Flags
Product Revision	Command and Firmware Release

Unlike the **TIM** command multiple **TM** function block can be used on one rung.

Refer to the **TIM** command for further details of timers. All other features are similar to the **TIM** command.

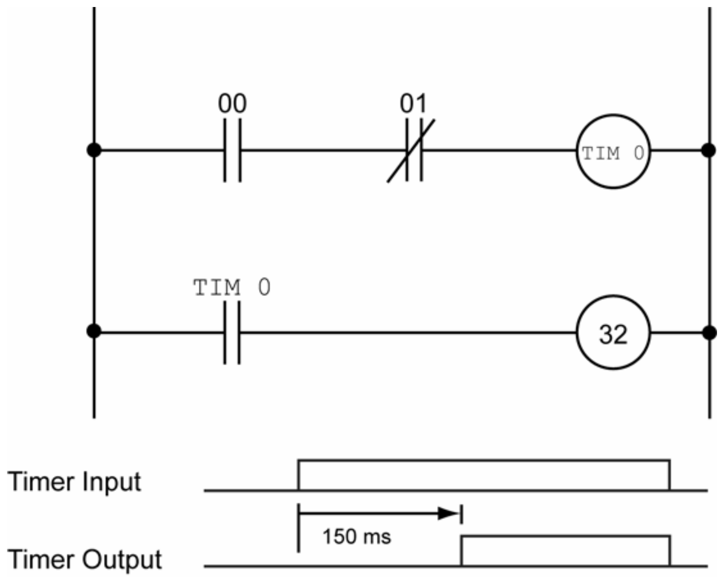
The following table outlines parameters and bit flags related to PLC timers. These are same for **TIM** and **TM**.

Timer	Preload	Count	Output	Input
0	P6660	P6661	Bit1552	Bit1553
1	P6676	P6677	Bit1584	Bit1585
2	P6692	P6693	Bit1616	Bit1617
3	P6708	P6709	Bit1648	Bit1649
4	P6724	P6725	Bit1680	Bit1681
5	P6740	P6741	Bit1712	Bit1713
6	P6756	P6757	Bit1744	Bit1745
7	P6772	P6773	Bit1776	Bit1777

Example 1 (TM command)

In this example, a normally open contact from relay 00 and a normally closed contact from relay 01 are connected in series to form a block. This block is then connected to the input coil of timer 0 which is set for 150 milliseconds. To bring out the state of the timer, a normally open contact from the timer is connected to the coil of relay 32.

The following illustrates a PLC timer:



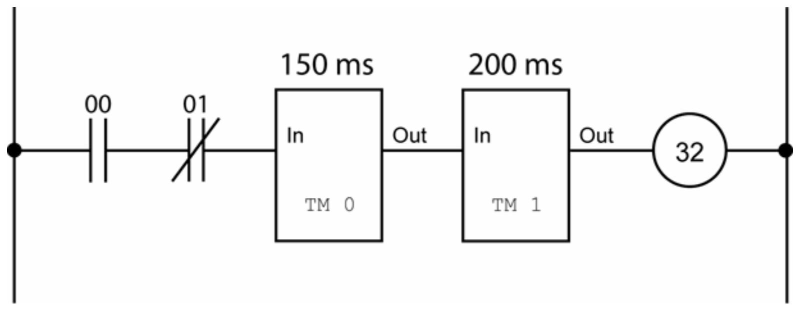
The following PLC code fragment implements the ladder logic shown above. Lines 100 and 110 create a new logic block. Line 120 connects the block to timer 0 and sets the timer to 150 milliseconds. Lines 130 and 140 connect the timer output to relay 32.

```
100 LD 00
110 AND NOT 01
120 TIM 0 150
130 LD TIM 0
140 OUT 32
```

Example 2 (TM command)

```
100 LD 00
110 AND NOT 01
120 TM 0 150
130 OUT 32
```

Example 3 (TM command)



```
100 LD 00
110 AND NOT 01
120 TM 0 150
130 TM 1 200
130 OUT 32
```

Index