PMC200-P2 Command

RERERENCE
MANUAL

Newport

Warranty

Newport Corporation warrants this product to be free from defects in material and workmanship for a period of 1 year from date of shipment. If found to be defective during the warranty period, the product will either be repaired or replaced at Newport's option.

To exercise this warranty, write or call your local Newport office or representative, or contact Newport headquarters in Irvine, California. You will be given prompt assistance and return instructions. Send the instrument, transportation prepaid, to the indicated service facility. Repairs will be made and the instrument returned, transportation prepaid. Repaired products are warranted for the balance of the original warranty period, or at least 90 days.

This warranty does not apply to defects resulting from modification or misuse of any product or part. This warranty also does not apply to fuses, batteries, or damage from battery leakage.

This warranty is in lieu of all other warranties, expressed or implied, including any implied warranty of merchantability or fitness for a particular use. Newport Corporation shall not be liable for any indirect, special, or consequential damages.

First Printing (July) 1989

© 1987 Newport Corporation
1791 Deere Avenue
Irvine, CA 92606
(949) 863-3144

P/N 17116-02 Rev. D2
IN-11896 (01-99)

Section 1.0
Introduction

The PMC200-P (GPIB) IEEE Interface Option is based upon the IEEE Standard 488.1 hardware standard and the IEEE 488.2 software standard that defines a standard digital interface for programmable instrumentation. This standard presents a well-defined instrumentation interface which simplifies the integration of instruments and computers into a system. Its popularity is due to its high data transfer rates, versatility and an unambiguous structure of codes, formats, protocols and common commands.

There are two command types; device independent ("common") commands, and device dependent commands. The common command type, also known as "bus commands", have the same meaning to all devices and are defined as part of the IEEE 488.2 Standard. All common commands start with an asterisk (*****). Commands that do not start with an asterisk are unique to the PMC200-P. Device dependent commands have been defined by Newport Corporation.

This manual provides detailed information and examples of all the commands for use with the PMC200-P IEEE-488 Interface Option. It is intended to be used as a reference document, to aid the user in using the remote control functions, via RS-232 or GPIB, of the PMC200-P.

The exact mnemonics and the syntax for all the valid commands are listed along with their required or optional parameters. If messages are returned to the controller, such as required by the query commands, the message structure is explained. The examples given under the command descriptions present some typical usage for the command.

Software registers are accessible by the user for aiding in communications and status reporting. The Status Byte and the Standard Event Registers, along with their enabling masks, are determined by the standard. The Device Event Register is unique to the PMC200-P. A graphical representation of these registers is presented in the appendices of this reference document.

Commands and queries may be sent either through the GPIB or the RS-232C port. The RS-232C port is configured to be used with IBM PC compatibles, using XON/XOFF handshaking for flow control. No other flow control method is currently supported by the PMC200-P. Please note that not all GPIB common commands are legal for RS-232C communication.

The RS-232C port is very useful for learning the command structure, as RS-232C communications are designed for interactive use. When input echoing is enabled, output is usually in the form of textual messages, and messages are returned to the remote host without polling. The GPIB port provides faster data transfer, but command errors are more difficult to detect unless the service request mechanism is implemented in the controller.

1.1 How This Manual is Organized

This manual is current for rev. 3.0 firmware and later. The firmware revision number is displayed at power-up or may be obtained via the *IDN? query. Contact Newport regarding firmware upgrades to earlier units.

The following section comments on the use of the PMC200-P RS-232C port. After that is a section that divides the commands into groups of similar actions, followed by a detailed description of each command in alphabetical order.

The Appendix includes the following important information:

- the definitions of the <*string*> data type. There are five types.
- the definitions of the <*number*> data type. There are four types.
- detailed information on errors.
- detailed information on the status reporting system.
- the register byte descriptions.
- sample RS-232 program in GWBASIC.
- sample GPIB program in QuickBASIC.

1.2 RS232C Communication

The PMC200-P may accept commands over the GPIB port or the RS-232C port, but not at the same time; only the controlling port can accept commands. Queries can be performed at any time through either port. The queried port need not be the controlling port. In most cases, the same commands may be sent from any port and will always be executed alike. If a command operates differently from the RS-232C port than from the GPIB port, the difference is discussed in the command description.

When the "Input Echo Mode" for the RS-232C port is enabled, the RS-232C port is designed to be interactive. All characters sent to the PMC200-P and error messages will be echoed. The PMC200-P generates a '>' prompt for every line. As the user is entering commands, the line may be edited by using the backspace key (sending an ASCII binary 08 code) or by using the DEL key (sending an ASCII binary 127 code). Currently there is no line abort key.

When the "Input Echo Mode:" is disabled, the RS-232C port is not interactive. The characters sent to the PMC200-P and error messages are not returned immediately. The error messages can be accessed by sending a *ERR? command to the PMC200-P followed by reading the string output from the PMC200-P.

Command lines sent to the PMC200-P are terminated by either a carriage return (sending an ASCII binary 13 code) or by a line feed (sending an ASCII binary 10 code).

When data is returned to the user terminal or remote host over the RS-232C port, all lines are terminated with a carriage return and line feed.

1.3 RS232 Quick Start using GWBASIC

For the IBM PC/AT or compatible

Important: Before proceeding with this quick start read the PMC200-P operators manual.

To Get a Quick Start do the following:

1. Connect a standard RS-232 cable from the PMC200-P RS-232 port to the computer's COM1 serial port.
2. Follow the instructions in section 2 of the PMC200-P operators manual to set-up the PMC200-P for RS-232 port control. Leave the baudrate at 9600, the parity at none, the databits at 8, and the stopbits at 1. Choose 'disabled' at the INPUT ECHO MODE: option. (see Operators Manual 2.8)
3. Boot up your computer and go to the directory that has GWBASIC in it. Start GWBASIC and type in the example program listed in appendix 4.8.
4. Run the example program. The program will clear the screen, write a header to the screen, and then wait for you to enter commands.
5. Type the command: POS?

If the PMC200-P is set-up properly, the program will send the motion query to the PMC200-P, read the position returned, and print it on the screen. You should see something like

```
00.0000,00.0000
```

on the screen.

NOTE: If you get timeout errors, check the cable connections and make sure the PMC200-P is set-up properly.

The RS-232 port can communicate with a dumb terminal or a personal computer running any of the many communications programs available (for example Mirror, White Knight and PC-TALK). You may also control your PMC200-P using a personal computer running high level programming languages such as Quick BASIC or C and lab automation software like LabWindows™ and LabVIEW™.

Section 2.0
Command Groups

The PMC200-P commands are grouped below in two categories: device independent common commands and device dependent commands. The common commands are commands that are specified by the IEEE 488.2 standard to work with instruments that support an IEEE 488.1 (also known as GPIB, HPIB or IEC-625) interface. For example, the *CLS command clears a device. The device dependent commands are specific to the PMC200-P. For example, the MOVE command causes the translator stage to move to a specified location.

The tables below shows both the common command set and the device dependent command set. It lists the mnemonic, which is the actual command, the function name of the command, and the group that the command belongs to. Commands belong to the same group when they are functionally similar.

Supported IEEE 488.2 Device Independent Commands

Command Group		**Description**
*IDN?	Identification Query	System Data
*RST	Reset	Internal Operations
*TST?	Self Test	Internal Operation
*OPC	Operation Complete (GPIB only)	Synchronization
*OPC?	Operation Complete Query	Synchronization
*WAI	Wait to Continue	Synchronization
*DMC	Define Macro	Macros
*EMC	Enable Macro	Macros
*EMC?	Enable Macro Query	Macros
*GMC?	Get Macro Contents	Macros
*LMC?	Learn Macro Query	Macros
*PMC	Purge Macro	Macros
*IST?	Individual Status Query (GPIB only)	Parallel Poll
*PRE	Parallel Poll Enable (GPIB only)	Parallel Poll
*PRE?	Parallel Poll Enable Query (GPIB only)	Parallel Poll
*CLS	Clear Status	Status and Event
*ERR?	Error Query	Status and Event
*ESE	Standard Event Status Enable	Status and Event
*ESE?	Standard Event Enable Query	Status and Event
*ESR?	Standard Event Register Query	Status and Event
*SRE	Service Request Enable (GPIB only)	Status and Event
*SRE?	Service Request Enable Query (GPIB only)	Status and Event
*STB?	Read Status Byte	Status and Event

IEEE 488.2 PMC200-P Device Dependent Commands

EVENT?	Device Event Register Query	Status & Event
EVENTEN	Device Event Enable Register	Status & Event
EVENTEN?	Device Event Enable Register Query	Status & Event
RMC	Remove Individual Macro Contents	Macro
ACTTYP	Define Actuator Type	System Setup
ALARM	Audio Alarm Enable/Disable	System Setup
ECHO	Enable RS-232 echo mode	System Setup
MOTION	Define Actuator Motion Type	System Setup
MTRTYP	Define Actuator Motor Type	System Setup
SYSDEF	System Defaults	System Setup
TOGPIB	Assign control to GPIB Port	System Setup
TO232	Assign control to RS-232C Port	System Setup

Command	Description	Group
UNITS	Units of Measurement	System Setup
ACTTYP?	Actuator Type Query	System Setup Query
ALARM?	Audio Alarm Status Query	System Setup Query
ECHO?	RS-232 echo mode query	System Setup Query
MOTION?	Actuator Motion Type Query	System Setup Query
MTRTYP?	Actuator Motor Type Query	System Setup Query
UNITS?	Units of Measurement Query	System Setup Query
ACL	Set Acceleration	Motion Control
ACL1	Set Axis 1 Acceleration	Motion Control
ACL2	Set Axis 2 Acceleration	Motion Control
BACKLASH	Actuator Backlash Factor	Motion Control
CPLRAT	Define Actuator Coupling Ratio	Motion Control
DISPDIR	Display Direction	Motion Control
PREPOS	Preset Actuator Position	Motion Control
SCALE	Scale Factor	Motion Control
ZERO	Set Origin	Motion Control
ZERO1	Set Axis 1 Origin	Motion Control
ZERO2	Set Axis 2 Origin	Motion Control
ACL?	Acceleration Query	Motion Control
ACL1?	Axis 1 Acceleration Query	Motion Control
ACL2?	Axis 2 Acceleration Query	Motion Control
BACKLASH?	Actuator Backlash Factor Query	Motion Control Query
CPLRAT?	Actuator Coupling Ratio Query	Motion Control Query
DISPDIR?	Display Direction Query	Motion Control Query
SCALE?	Scale Factor Query	Motion Control Query
HOME	Move to Origin	Motion Execute
HOME1	Move Axis 1 to Origin	Motion Execute
HOME2	Move Axis 2 to Origin	Motion Execute
JOG	Move to Relative Destination	Motion Execute
JOG1	Move Axis 1 to Relative Destination	Motion Execute
JOG2	Move Axis 2 to Relative Destination	Motion Execute
MOVE	Move to Destination	Motion Execute
MOVE1	Move Axis 1 to Destination	Motion Execute
MOVE2	Move Axis 2 to Destination	Motion Execute
RUN	Run Axis 1 & Axis 2	Motion Execute
RUN1	Run Axis 1	Motion Execute
RUN2	Run Axis 2	Motion Execute
STOP	Stop Motion	Motion Execute
STOP1	Stop Axis 1 Motion	Motion Execute
STOP2	Stop Axis 2 Motion	Motion Execute
VEL	Set Axis 1 & Axis 2 Maximum Velocity	Motion Execute
VEL1	Set Axis 1 Maximum Velocity	Motion Execute
VEL2	Set Axis 2 Maximum Velocity	Motion Execute
AVEL?	Actual Velocity Query	Motion Query
AVEL1?	Axis 1 Actual Velocity Query	Motion Query
AVEL2?	Axis 2 Actual Velocity Query	Motion Query
JOG?	Relative Motion Query	Motion Query
JOG1?	Axis 1 Relative Motion Query	Motion Query
JOG2?	Axis 2 Relative Motion Query	Motion Query
MOVE?	Move Destination Query	Motion Query
MOVE1?	Axis 1 Move Destination Query	Motion Query
MOVE2?	Axis 2 Move Destination Query	Motion Query
POS?	Current Position Query	Motion Query
POS1?	Axis 1Current Position Query	Motion Query
POS2?	Axis 2 Current Position Query	Motion Query

RUN?	Run Velocity Query	Motion Query
RUN1?	Axis 1 Run Velocity Query	Motion Query
RUN2?	Axis 2 Run Velocity Query	Motion Query
VEL?	Axis. 1 & Axis 2 Maximum Velocity Query	Motion Query
VEL1?	Axis 1 Maximum Velocity Query	Motion Query
VEL2?	Axis 2 Maximum Velocity Query	Motion Query
SMPL	Set HCTL-1100 Sampling Frequency	Advanced Parameters
SMPL?	Query HCTL-1100 Sampling Frequency	Advanced Parameters
GAINF	Set HCTL-1100 Gain Parameter	Advanced Parameters
GAINF?	Query HCTL-1100 Gain Parameter	Advanced Parameters
POLEF	Set HCTL-1100 Pole Parameter	Advanced Parameters
POLEF?	Query HCTL-1100 Pole Parameter	Advanced Parameters
ZEROF	Set HCTL-1100 Zero Parameter	Advanced Parameters
ZEROF?	Query HCTL-1100 Zero Parameter	Advanced Parameters
TIME	Set Current Time	Timer
TIME?	Current Time Query	Timer Query

Section 3.0 Command Descriptions

Following is an alphabetical listing of the entire PMC200-P command set. In the command descriptions, several abbreviations are used. The abbreviations are listed below:

<EOI> End of Identify.

An IEEE 488.1 signal sometimes sent with the <NL> character.

<IST> Individual status

Generated by the *PRE command, and used as part of the conditional generation for a parallel poll.

<NL> New Line.

An ASCII decimal 10 byte.

<CR> Carriage Return

An ASCII decimal 13 byte.

<SRQ> Service Request.

A device generates a <SRQ> to tell the controller that a serial poll is needed.

The command parameters are all shown in <*italics*>. If a parameter is optional, then it will be surrounded by square brackets [<*italics*>]. If a command has more than one parameter, the parameters are separated by commas.

Semicolons separate commands within a single transmission. A transmission is ended when the <EOI> is active on a data byte, or the <NL> character is received.

Whitespace is optional in between commands and between parameters. Whitespace is any character with a binary value less than or equal to an ASCII space character (except the <NL> character).

Numeric parameter types are denoted by <*number*>. Numeric parameters are passed and returned as the actual ASCII characters in the string representation of the number. String parameters or ASCII data parameters types are denoted by <*string*>. When a parameter value is optional, it will be enclosed in square brackets. For example, [<*number*>] represents an optional numeric parameter value, while <*number*> represents a required parameter value.

The PMC200-P accepts numeric values in Decimal, Scientific, Octal, Binary, or Hexadecimal format (see Appendix 4.2). The parameter is automatically truncated if necessary. For example, the TIME command requires three parameters: Hours, Minutes, and am/pm. The last one is optional, so TIME 11:31.1am would set the time to 11:31 am. The 31.1 is truncated to 31. The command TIME 11:3.1E1am would have the same effect. In order to denote the fact that the last parameter is optional, the syntax for TIME is given as TIME <hours>:<minutes>[<am_pm>].

All motion units are expressed in inches, millimeters, degrees, and milliradians as specified by the user. Velocity is defined in *units*/sec and acceleration is defined in *units*/sec^2.

When commands are received from the RS-232C port, either a <CR> or a <NL> is treated as the end of command sequence character. <NULL> characters are ignored.

When commands are received from the GPIB port, either an <EOI> with a data byte or a <NL> is treated as the end of sequence character. The preferred method is for the <EOI> line to be active with the <NL> character.

Each command sequence must be terminated by the end of sequence condition (<NL> or <EOI>). A command sequence may be a single command or a group of commands. Each command in a group of commands is separated by a semicolon and white space is ignored. Semicolons may be part of a <*string*> data item and will not terminate the command associated with that string (see the definition of <*string*>).

The syntax of the data returned by the PMC200-P follows that shown by the individual query commands. In general all returned data is terminated by a <CR> and <NL> characters. When queries are returned over the GPIB port, the <EOI> line is always active when the <NL> character is transmitted.

3.1 ACL Set Axis 1 & 2 Acceleration/ Deceleration

Syntax: ACL [<*axis 1_acl*>][,<*axis 2_acl*>]

Parameters:

<*axis 1_acl*> converts to a floating point <*number*> that represents the axis 1 actuator acceleration and deceleration in units per second^2 (units/sec^2). If <*axis 1_acl*> is missing, then the current axis 1 actuator acceleration value is not changed.

<*axis 2_acl*> converts to a floating point <*number*> that represents the axis 2 actuator acceleration and deceleration in units per second^2 (units/sec^2). If <*axis 2_acl*> is missing, then the current axis 2 actuator acceleration value is not changed.

Function:

This command establishes the slope of the leading and trailing edges of the trapezoidal velocity profile for point-to-point moves (see MOVE, HOME, and JOG). The values established by issuing this command are nominal; actual accelerations and decelerations are dependent on load. In addition, the actual acceleration and deceleration may differ from each other when the effective load in one direction is different from that in the other direction, as is the case with vertical loading or when heavy return springs are used.

Acceleration Defaults

850A	2.000mm/sec
850B	1.000mm/sec
850B-LS	0.175mm/sec
850B-HS	60.00mm/sec
495	40.00deg/sec
496	20.00deg/sec

Returns: NONE

Example:

Send: ACL 2.0,2.0<*NL*>

Sets the approximate acceleration of axes 1 and 2 to 2.0 <units>/sec^2.

Related Commands: ACL?, ACL1, ACL1?, ACL2, and ACL2?

3.2 ACL? Axis 1 & 2 Acceleration/ Deceleration Query

Syntax: ACL?

Parameters: NONE

Function:

This command returns the acceleration set by the an ACL command which defines the slope of the leading and trailing edges of the trapezoidal velocity profile for point-to-point moves. The units of the return value are <units>/sec^2. The values returned are nominal; actual accelerations and decelerations are dependent on load. In addition, the actual acceleration and deceleration may differ from each other when the effective load in one direction is different from that in the other direction, as is the case with vertical loading or when heavy return springs are used.

Returns: *<axis 1_acl>,<axis 2_acl><NL>*

Example:

```
Send:      ACL?<NL>
Receive:   2.0000,2.0000<NL>
```

Queries the acceleration set for axes 1 and 2.

Related Commands: ACL, ACL1, ACL1?, ACL2, and ACL2?

3.3 ACL1 Set Axis 1 Acceleration/ Deceleration

Syntax: ACL1 *<axis 1_acl>*

Parameters:

<axis 1_acl> converts to a floating point *<number>* that represents the axis 1 actuator acceleration and deceleration in units per second^2 (units/sec^2). If *<axis 1_acl>* is missing, then the current axis 1 actuator acceleration value is not changed.

Function:

This command establishes the slope of the leading and trailing edges of the trapezoidal velocity profile for point-to-point moves (see MOVE, HOME, and JOG). The values established by issuing this command are nominal; actual accelerations and decelerations are dependent on load. In addition, the actual acceleration and deceleration may differ from each other when the effective load in one direction is different from that in the other direction, as is the case with vertical loading or when heavy return springs are used.

Acceleration Defaults

850A	2.000mm/sec
850B	1.000mm/sec
850B-LS	0.175mm/sec
850B-HS	60.00mm/sec
495	40.00deg/sec
496	20.00deg/sec

Returns: NONE

Example:

```
Send:      ACL1 2.0<NL>
```

Sets the approximate acceleration of axis 1 to 2.0 <units>/sec^2.

Related Commands: ACL, ACL?, ACL1?, ACL2, and ACL2?

3.4 ACL1? Axis 1 Acceleration/ Deceleration Query

Syntax: ACL1?

Parameters: NONE

Function:

This command returns the acceleration set by the an ACL command which defines the slope of the leading and trailing edges of the trapezoidal velocity profile for point-to-point moves. The units of the return value are <units>/sec^2. The values returned are nominal; actual accelerations and decelerations are dependent on load. In addition, the actual acceleration and deceleration may differ from each other when the effective load in one direction is different from that in the other direction, as is the case with vertical loading or when heavy return springs are used.

Returns: <*axis 1_acl*><*NL*>

Example:

Send:	ACL1?<*NL*>
Receive:	2.0000<*NL*>

Queries the acceleration set for axis 1.

Related Commands: ACL, ACL?, ACL1, ACL2, and ACL2?

3.5 ACL2 Set Axis 2 Acceleration/ Deceleration

Syntax: ACL2 <*axis 2_acl*>

Parameters:

<*axis 2_acl*> converts to a floating point <*number*> that represents the axis 2 actuator acceleration and deceleration in units per second^2 (units/sec^2). If <*axis 2_acl*> is missing, then the current axis 2 actuator acceleration value is not changed.

Function:

This command establishes the slope of the leading and trailing edges of the trapezoidal velocity profile for point-to-point moves (see MOVE, HOME, and JOG). The values established by issuing this command are nominal; actual accelerations and decelerations are dependent on load. In addition, the actual acceleration and deceleration may differ from each other when the effective load in one direction is different from that in the other direction, as is the case with vertical loading or when heavy return springs are used.

Acceleration Defaults

850A	2.000mm/sec
850B	1.000mm/sec
850B-LS	0.175mm/sec
850B-HS	60.00mm/sec
495	40.00deg/sec
496	20.00deg/sec

Returns: NONE

Example:

Send:	ACL2 2.0<NL>

Sets the approximate acceleration of axis 2 to 2.0 <units>/sec^2.

Related Commands: ACL, ACL?, ACL1, ACL1?, and ACL2?

3.6 ACL2? Axis 2 Acceleration/ Deceleration Query

Syntax: ACL2?

Parameters: NONE

Function:

This command returns the acceleration set by the an ACL command which defines the slope of the leading and trailing edges of the trapezoidal velocity profile for point-to-point moves. The units of the return value are <units>/sec^2. The values returned are nominal; actual accelerations and decelerations are dependent on load. In addition, the actual acceleration and deceleration may differ from each other when the effective load in one direction is different from that in the other direction, as is the case with vertical loading or when heavy return springs are used.

Returns: *<axis 2_acl><NL>*

Example:

Send:	ACL2?*<NL>*
Receive:	2.0000*<NL>*

Queries the acceleration set for axis 2.

Related Commands: ACL, ACL?, ACL1, ACL1?, and ACL2

3.7 ACTTYP Define Actuator Type

Syntax: ACTTYP [<*axis 1_act*>][,<*axis 2_act*>]

Parameters:

<*axis 1_act*> is of type <*string*>, and represents the actuator type attached to axis 1. As some of the actuator types appear to the system as integers, the string should be quoted. The currently defined values for the parameter fields are given in the table below.

<*axis 2_act*> is of type <*string*>, and represents the actuator type attached to axis 2. As some of the actuator types appear to the system as integers, the string should be quoted. The currently defined values for the parameter fields are given in the table below.

Defined Actuator Types

495	Newport 495 rotary actuator
496	Newport 496 rotary actuator
850	Newport 850 or 850A linear actuator
850B	Newport 850B Series linear actuator
850B-LS	Newport 850B Low Speed linear actuator
850B-HS	Newport 850B High Speed linear actuator
SPECIAL	non-standard linear actuator
NONE	no actuator attached

Function:

This command informs the PMC200-P what actuators, if any, are currently attached to the PMC200-P on axis 1 and axis 2. The default for each axis is “NONE”. With the exception of special linear actuators, the PMC200-P will modify all current system parameters to match the defined actuator when this command is received. You must inform the PMC200-P what devices are connected by this command or by its front panel menu set-up. It does not “know” when a device is connected or what kind of device it is.

The ACTTYP command is not case-sensitive with respect to parameters. For example, “SPECIAL” is the same as “special”. All characters in the parameter string are significant.

When a special actuator is defined, the MTRTYP and CPLRAT commands should be used to define the actuator motor type and coupling ratio (gear ratio), respectively, of the special actuator. Failure to do so can result in incorrect actuator motion and returned position and actual velocity values.

If the ACTTYP command defines no actuator for any axis, the PMC200-P will deactivate that axis. Even if an actuator is actually attached to that axis, the PMC200-P will still deactivate the axis.

Returns: NONE

Examples:

```
Send:      ACTTYP "495","850"<NL>
Send:      ACTTYP?<NL>
Receive:   "495","850"<NL>
```

The above example sets all axis 1 operational parameters for a 495-type rotary actuator, and axis 2 for an 850-type actuator.

```
Send:      ACTTYP ,"SPECIAL"<NL>
Send:      MTRTYP ,"1624"<NL>
Send:      CPLRAT ,1600<NL>
```

The above example sets axis 2 actuator type to “special”, and the MTRTYP command sets the actuator motor type to “1624”, and finally the CPLRAT command sets the gear ratio for 1600 quadrature counts per mm. The axis 1 actuator was not affected.

Related Commands: ACTTYP?, MOTION, MOTION?, MTRTYP, MTRTYP?, CPLRAT, CPLRAT?, GAINF, POLEF, ZEROF

3.8 ACTTYP? Actuator Type Query

Syntax: ACTTYP?

Parameters: NONE

Function:

This query returns the defined actuators, if any, that the PMC200-P has been informed are currently attached to the PMC200-P on axis 1 and axis 2. The default for each axis is "NONE".

If the ACTTYP command defines no actuator for any axis, the PMC200-P will return "NONE" for that axis. Even if an actuator is actually attached to that axis, the PMC200-P will still return "NONE" for the axis.

Returns: *<axis 1_act>*,*<axis 2_act>*

<axis 1_act> is of type *<string>*, and represents the actuator type attached to axis 1. The currently defined values for the parameter fields are given in the table below.

<axis 2_act> is of type *<string>*, and represents the actuator type attached to axis 2. The currently defined values for the parameter fields are given in the table below.

Defined Actuator Types

495	Newport 495 rotary actuator
496	Newport 496 rotary actuator
850	Newport 850 linear actuator
850B	Newport 850B Series linear actuator
850B-LS	Newport 850B Low Speed linear actuator
850B-HS	Newport 850B High Speed linear actuator
SPECIAL	non-standard linear actuator
NONE	no actuator attached

Examples:

Send:	ACTTYP "495","850"*<NL>*
Send:	ACTTYP?*<NL>*
Receive:	"495","850"*<NL>*

The above example sets all axis 1 operational parameters for a 495-type rotary actuator, and axis 2 for an 850-type actuator.

Send:	ACTTYP ,"SPECIAL"
Send:	MTRTYP ,"1624"
Send:	CPLRAT ,1600

The above example sets axis 2 actuator type to "special", and the MTRTYP command sets the actuator motor type to "1624", and finally the CPLRAT command sets the gear ratio for 1600 quadrature counts per mm. The axis 1 actuator was not affected.

Related Commands: ACTTYP, MOTION, MOTION?, MTRTYP, MTRTYP?, CPLRAT, CPLRAT?, GAINF, POLEF, ZEROF

3.9 ALARM Audio Alarm Enable/ Disable

Syntax: ALARM <*alarm_state*>

Parameters:

<*alarm_state*> is of type <*string*>, and is used to enable or disable the PMC200-P's audio alarm.

Function:

This command is used to control the state of the PMC200-P's audio alarm. The audio alarm is a piezo crystal device internal to the PMC200-P. The audio alarm gives a non-visual cue to the user that a limit condition has been reached on an actuator, if enabled.

The ALARM command is not case-sensitive with respect to the parameter content. For example, "ENABLE" is the same as "enable". Only the first character in the parameter string is significant. The only supported strings must start with either "E" (enable) or"D" (disable).

Returns: NONE

Examples:

```
Send:        ALARM "D"<NL>
```

This will disable the alarm. Note that only one character was used in the string.

Related Commands: ALARM?

3.10 ALARM? Audio Alarm Enable/ Disable Query

Syntax: ALARM?

Parameters: NONE

Function:

This query returns the current state of the PMC200-P's audio alarm. The audio alarm is a piezo crystal device internal to the PMC200-P. The audio alarm gives a non-visual cue to the user that a limit condition has been reached on an actuator, if enabled.

Returns: <*alarm_state*>

The following values are defined for the ALARM? query:

"ENABLED"
"DISABLED"

Examples:

```
Send:        ALARM?<NL>
Receive:     "DISABLED"<NL>
```

This response shows the alarm is disabled.

Related Commands: ALARM

3.11 AVEL? Actual Velocity Query

Syntax: AVEL?

Parameters: NONE

Function:

This command returns the actual instantaneous velocities of the actuators, when motion is due to a RUN, RUN1 or RUN2 command. Velocity is given in units per second. Actual velocity may be less than requested velocity, due to loading or actuator limitations.

Returns: *<axis 1_vel>*,*<axis 2_vel><NL>*

<axis 1_vel> is a floating point *<number>* that represents the axis 1 actuator velocity in units per second (units/sec). This is the actual velocity at the time the AVEL? query was executed. If the actuator motion is not due to a RUN or RUN1 command, the value returned will be 0. The factory default value is 0.0 units per second. The returned value is in standard signed floating point format.

<axis 2_vel> is a floating point *<number>* that represents the axis 2 actuator velocity in units per second (units/sec). This is the actual velocity at the time the AVEL? query was executed. If the actuator motion is not due to a RUN or RUN2 command, the value returned will be 0. The factory default value is 0.0 units per second. The returned value is in standard signed floating point format.

Examples:

Send:	RUN 10.1,1.0*<NL>*
Send:	AVEL?*<NL>*
Receive:	001.100,001.000*<NL>*

Actual velocity is less than requested velocity on axis 1, due to actuator loading or limitations.

Related Commands: RUN, RUN1, RUN2, AVEL1?, AVEL2?

3.12 AVEL1? Axis 1 Actual Velocity Query

Syntax: AVEL1?

Parameters: NONE

Function:

This command returns the actual instantaneous velocity of the axis 1 actuator, when motion is due to a RUN or RUN1 command. Velocity is given in units per second. Actual velocity may be less than requested velocity, due to loading or actuator limitations.

Returns: *<axis 1_vel><NL>*

<axis 1_vel> is a floating point *<number>* that represents the axis 1 actuator velocity in units per second (units/sec). This is the actual velocity at the time the AVEL1? query was executed. If the actuator motion is not due to a RUN or RUN1 command, the value returned will be 0. The factory default value is 0.0 units per second. The returned value is in standard signed floating point format.

Examples:

```
Send:       RUN1 10.1<NL>
Send:       AVEL1?<NL>
Receive:    1.100<NL>
```

Actual velocity is less than requested velocity, due to actuator loading or limitations.

```
Send:       RUN1 -.354<NL>
Send:       AVEL1?<NL>
Receive:    -.35400<NL>
```

Related Commands: RUN, RUN1, RUN2, AVEL?, AVEL2?

3.13 AVEL2? Axis 1 Actual Velocity Query

Syntax: AVEL2?

Parameters: NONE

Function:

This command returns the actual instantaneous velocity of the axis 2 actuator, when motion is due to a RUN or RUN2 command. Velocity is given in units per second. Actual velocity may be less than requested velocity, due to loading or actuator limitations.

Returns: *<axis2_vel><NL>*

<axis 2_vel> is a floating point *<number>* that represents the axis 2 actuator velocity in units per second (units/sec). This is the actual velocity at the time the AVEL2? query was executed. If the actuator motion is not due to a RUN or RUN2 command, the value returned will be 0. The factory default value is 0.0 units per second. The returned value is in standard signed floating point format.

Examples:

```
Send:      RUN2 10.1<NL>
Send:      AVEL2?<NL>
Receive:   1.100<NL>
```

Actual velocity is less than requested velocity, due to actuator loading or limitations.

```
Send:      RUN2 -.354<NL>
Send:      AVEL2?<NL>
Receive:   -.35400<NL>
```

Related Commands: RUN, RUN1, RUN2, AVEL?, AVEL1?

3.14 BACKLASH Actuator Backlash Factor

Syntax: BACKLASH [<*axis 1_arg*>][,<*axis 2_arg*>]

Parameters:

<*axis 1_arg*> is of type <*number*>, and is interpreted as an unsigned floating point number. <*axis 1_arg*> sets the backlash factor, in currently defined units, for the actuator attached to axis 1.

<*axis 2_arg*> is of type <*number*>, and is interpreted as an unsigned floating point number. <*axis 2_arg*> sets the backlash factor, in currently defined units, for the actuator attached to axis 2.

NOTE: The backlash correction factor always has units of mm for linear devices and degrees for rotation stages, regardless of the display units selected. That is, if you are using an 850B and have selected inches as the display units, the backlash correction factor should still be entered in mm.

Function:

This command defines the backlash compensation factor for each individual actuator and their "downstream" devices. Backlash is the amount of positional error that is introduced when changing direction of actuator travel due to gear play in the actuator and other mechanical losses. The PMC200-P system default value for backlash is 0.0 for all actuators, which effectively means backlash compensation is turned off.

Along with friction/stiction issues, backlash is the dominant motion error contributing to bi-directional repeatability performance. When an actuator changes direction, the PMC200-P has the capability of adding-in a drivetrain input equal to the device's backlash factor, greatly improving the device's repeatability. The backlash factor for all Newport actuators is invariably a very small distance, typically 5µm or so for 850-B actuators.

The backlash factor for each actuator should be determined by measuring the smallest motion that can be performed after a change of direction. For example, an actuator with 5 µm of backlash will show no motion when incremented forward and backward in 4 µm commanded jogs, but at 5 µm will begin to show small motions. These measurements can be made with the aid of a microscope or loupe with backlash compensation set to zero for the duration of the test.

When a new actuator is defined for a given axis using the ACTTYP command, the backlash factor for that axis is reset to zero.

Returns: NONE

Examples:

```
Send:      BACKLASH .001,.0033<NL>
Send:      BACKLASH?<NL>
Receive:   000.001,00.0033<NL>
```

The above example sets backlash factoring for axis 1 to .001 units, and sets backlash factoring for axis 2 to .0033 units.

Related Commands: BACKLASH?

3.15 BACKLASH? Actuator Backlash Factor Query

Syntax: BACKLASH?

Parameters: NONE

Function:

This query returns the defined backlash compensation factors for each axis. The units are always mm for linear devices and degrees for rotary stages, regardless of the display units chosen. Backlash is the amount of positional error that is introduced when changing direction of actuator travel due to gear play in the actuator. The PMC200-P system default value for backlash is 0.0 for both actuators.

See the discussion of the of the BACKLASH command for more information on how the backlash compensation factor is determined and used.

Note: When a new actuator is defined for a given axis using the ACTTYP command, the backlash factor for that axis is automatically set to zero.

Returns: *<axis 1_arg>,<axis 2_arg>*

<axis 1_arg> is of type *<number>*, and is interpreted as an unsigned floating point number. *<axis 1_arg>* represents the backlash factor, for the actuator attached to axis

<axis 2_arg> is of type *<number>*, and is interpreted as an unsigned floating point number. *<axis 2_arg>* represents the backlash factor, for the actuator attached to axis

Examples:

```
Send:      ACTTYP “495”,”850"<NL>
Send:      BACKLASH?<NL>
Receive:   000.000,000.000<NL>
```

In this example, backlash is automatically set to 0.0 when the ACTTYP command is used to define new actuators.

```
Send:      BACKLASH ,0.001<NL>
Send:      BACKLASH?<NL>
Receive:   000.000,000.001<NL>
```

The BACKLASH command is used here to define the axis 2 actuator as having a backlash factor of .001 units. The axis 1 actuator was not affected.

Related Commands: BACKLASH

3.16 *CLS Clear Status

Syntax: *CLS

Parameters: NONE

Function:

This common command clears all the status registers. The registers cleared are: Status Register, Standard Event Register, and Device Event Register. The *CLS common command also removes any errors from the Error Queue. The *CLS common command is mainly used to clear the bit(s) that generated a request for serial or parallel poll. While *CLS clears the status byte, the Message Available (bit 4) of the status byte is not affected.

Returns: NONE

Examples:

Send:	*CLS<*NL*>
Send:	*ESR?; EVENT?; *STB?<*NL*>
Receive:	0<*NL*>
Receive:	0<*NL*>
Receive:	0<*NL*>

The *CLS command clears the Standard Event Status Register, the Device Event Register, and the Status Byte.

Related Commands: *ESR?, EVENT?

3.17 CPLRAT Define Actuator Coupling Ratio

Syntax: CPLRAT [<*axis 1_ratio*>][,<*axis 2_ratio*>]

Parameters:

<*axis 1_ratio*> is of type <*number*>, and is evaluated as an unsigned floating point value. <*axis 1_ratio*> defines the number of quadrature counts per unit for the actuator. Unit is defined as millimeters for linear actuators, and degrees for rotary actuators.

<*axis 2_ratio*> is of type <*number*>, and is evaluated as an unsigned floating point value. <*axis 2_ratio*> defines the number of quadrature counts per unit for the actuator. Unit is defined as millimeters for linear actuators, and degrees for rotary actuators.

Function:

In order for the PMC200-P to compute accelerations, velocities, and position, the coupling ratio, or gear ratio, for an actuator is required. This is especially true in the case of special actuators being defined. The CPLRAT command allows the actuator gear ratio parameter to be changed.

In the case of standard 495, 496, and 850/850A actuators, the PMC200-P will automatically set the coupling ratio to the factory defined value when the actuator is selected for use. The CPLRAT command will override the factory default value.

When selecting special actuators, the CPLRAT command must be used to set the coupling ratio for the actuator, since it does not have a factory default defined within the PMC200-P. The coupling ratio usually is obtained from documentation received with the actuator. It may also be calculated by the following simple formula:

$$\text{Coupling ratio} = 20{,}000 * \frac{\text{Gearboxratio}}{262}$$

that is, for standard 850/850A actuators (which have 262:1 gear ratios) The coupling ratio is 20,000. In addition, the MTRTYP command must be used to set the correct motor type for the specific actuator.

A listing of available 850B motor/gearbox options follows:

Actuator	Motor	Gearbox	Coupling Ratio
850B-HS	1624	22:1	1,679
850B-LS	1516 (similar to 1616)	1,670:1	127,481
850B	1616	262:1	20,000

Returns: NONE

Examples:

```
Send:      ACTTYP "495","850"<NL>
Send:      CPLRAT ,1666.7<NL>
Send:      CPLRAT?<NL>
Receive:   2000,1666.7<NL>
```

The above example sets all axis 1 operational parameters for a 495-type rotary actuator, and axis 2 for an 850-type actuator. Then the coupling ratio for axis 2 is redefined to 1666.7 quadrature counts per millimeter. Note that the 495 actuator has a predefined coupling ratio.

```
Send:      ACTTYP ,"SPECIAL"
Send:      MTRTYP ,"1624"
Send:      CPLRAT ,1600
```

The above example sets axis 2 actuator type to “special”, and the MTRTYP command sets the actuator motor type to “1624”, and finally the CPLRAT command sets the gear ratio for 1600 quadrature counts per mm. The axis 1 actuator was not affected.

Related Commands: ACTTYP, ACTTYP?, MOTION, MOTION?, MTRTYP, MTRTYP?, CPLRAT?

3.18 CPLRAT? Actuator Coupling Ratio Query

Syntax: CPLRAT?

Parameters: NONE

Function:

This query returns the defined coupling ratio for each actuator axis. If an axis is defined as having no actuator, the last defined coupling ratio is returned. The coupling ratio for each actuator can be set by the CPLRAT command, or will be set automatically by the ACTTYP command. The coupling ratio for special actuators must be set using the CPLRAT command.

A listing of available 850B motor/gearbox options follows:

Actuator	Motor	Gearbox	Coupling Ratio
850B-HS	1624	22:1	1,679
850B-LS	1516 (similar to 1616)	1,670:1	127,481
850B	1616	262:1	20,000

Returns: <*axis 1_ratio*>,<*axis 2_ratio*>

<*axis 1_ratio*> is of type <*number*>, and is evaluated as an unsigned floating point value. <*axis 1_ratio*> defines the number of quadrature counts per unit for the actuator. Unit is defined as millimeters for linear actuators, and degrees for rotary actuators.

<*axis 2_ratio*> is of type <*number*>, and is evaluated as an unsigned floating point value. <*axis 2_ratio*> defines the number of quadrature counts per unit for the actuator. Unit is defined as millimeters for linear actuators, and degrees for rotary actuators.

Example:

Send:	ACTTYP “495”,”850"<*NL*>
Send:	CPLRAT?<*NL*>
Receive:	2000.00,20000.00<*NL*>

The above example sets all axis 1 operational parameters for a 495-type rotary actuator, and axis 2 for an 850-type actuator.

Related Commands: ACTTYP, ACTTYP?, MOTION, MOTION?, MTRTYP, MTRTYP?, CPLRAT

3.19 DISPDIR Display Direction

Syntax: DISPDIR [<*axis 1_dir*>][,<*axis 2_dir*>]

Parameters:

<*axis 1_dir*> is of type <*string*>, and is used to define the PMC200-P mapping of actuator motion to system display motion directions “forward” and “reverse”, for axis 1.

<*axis 2_dir*> is of type <*string*>, and is used to define the PMC200-P mapping of actuator motion to system directions “forward” and “reverse”, for axis 2.

The table below gives valid parameter strings for linear and rotary actuators. Parameter strings for linear and rotary actuators are not interchangeable.

String	Mapping	Motion
IN	forward = inward motion	Linear
OUT	forward = outward motion	Linear
CW	forward = clockwise motion	Rotary
CCW	forward = counter clockwise	Rotary

Function:

This command allows the user to map actuator motion to the PMC200-P’s display motion directions of forward and reverse. Forward and reverse mapping defaults are dependent on actuator motion type. Forward motion results in a positive delta for position. Reverse motion gives a negative delta for position.

Linear actuators have an immobile “body” housing the encoder and drive motor, and an armature or spindle that moves inward or outward from that body. The PMC200-P system default for linear actuators map “Forward = out”. In this case, outward motion of the actuator armature is initiated when a positive displacement or velocity is given in motion commands.

Rotary actuators have an immobile “body” and a platen that rotates clockwise or counter-clockwise. The PMC200-P system default for rotary actuators map “Forward = cw”. In this case clockwise rotation of the actuator platen is initiated when a positive displacement or velocity is given in motion commands.

Returns: NONE

Examples:

Send:	ACTTYP “495”,”850"<*NL*>
Send:	DISPDIR “CCW”<*NL*>
Send:	DISPDIR?<*NL*>
Receive:	“Forward = ccw”,”Forward = out”<*NL*>

The above example sets all axis 1 operational parameters for a 495-type rotary actuator, and axis 2 for an 850-type actuator. The DISPDIR command is used to map counter-clockwise movement to forward motion. The DISPDIR? query shows that the PMC200-P has mapped forward motion as counter-clockwise movement for the 495 actuator, and forward motion as outward movement for the 850 actuator.

Related Commands: DISPDIR?

3.20 DISPDIR? Display Direction Query

Syntax: DISPDIR?

Parameters: NONE

Function:

This query returns the defined mapping of actuator motion to the PMC200-P's display motion directions of forward and reverse. Forward and reverse mapping defaults are dependent on actuator motion type. Forward motion results in a positive delta for position. Reverse motion gives a negative delta for position.

Linear actuators have an immobile "body" housing the encoder and drive motor, and an armature or spindle that moves inward or outward from that body. The PMC200-P system default for linear actuators map "Forward = out". In this case, outward motion of the actuator armature is initiated when a positive displacement or velocity is given in motion commands.

Rotary actuators have an immobile "body" and a platen that rotates clockwise or counter-clockwise. The PMC200-P system default for rotary actuators map "Forward = cw". In this case clockwise rotation of the actuator platen is initiated when a positive displacement or velocity is given in motion commands.

Returns: <*axis 1_dir*>,<*axis 2_dir*>

<*axis 1_dir*> is of type <*string*>, and reflects the PMC200-P mapping of actuator motion to system display motion directions "forward" and "reverse", for actuator axis 1.

<*axis 2_dir*> is of type <*string*>, and reflects the PMC200-P mapping of actuator motion to system directions "forward" and "reverse", for actuator axis 2.

The table below gives valid return strings for linear and rotary actuators. Return strings for linear and rotary actuators are not interchangeable.

String	Mapping	Motion
Forward=in	forward = inward motion	Linear
Forward=out	forward = outward motion	Linear
Forward-cw	forward = clockwise motion	Rotary
Forward-ccw	forward = counter clockwise motion	Rotary

Examples:

```
Send:      DISPDIR?<NL>
Receive:   "Forward = cw","Forward = out"<NL>
```

The DISPDIR? query returns show that the axis 1 actuator movement is mapped to forward = clockwise movement, and the axis 2 actuator movement is mapped to forward = outward movement. Likewise, these return values show that he axis 1 actuator is declared to be a rotary actuator, and axis 2 is a linear actuator.

Related Commands: DISPDIR

3.21 *DMC Define Macro Contents

Syntax: *DMC <*name*>,<*commands*>

Parameters:

<*name*> is the name of the macro and is of type <*string*>. No case conversion is performed on alphabetic characters in macro names; this is, case is significant. The first character must be an alphabetic (A-Z, a-z). All subsequent characters must be from the following list:

0-9 A-Z _ ? :

The length of the name is not restricted, but only the first 16 characters are significant. An error is generated if <*name*> is already defined as a macro. An error is also generated if the macro is redefining an existing command, and <*name*> is a non-quoted string.

<*commands*> is of type <*string*>. The string is a list of commands to execute, when <*name*> is sent as a command.

Function:

This command stores a command sequence which is executed when the macro name is received. The command sequence may contain any number of commands, each separated by semicolons. Other macro names are allowed in the command string. A macro may have the same name as a device specific command, whereby preventing the device specific command from being executed. Macro expansion must be enabled by using the *EMC common command. The factory default is no macros defined, and macro expansion disabled.

<*name*> must be unique, otherwise an error is generated. If you wish to redefined an existing macro, you have two choices. First is to use the RMC device dependent command to first remove the macro and its contents. Then you may use the *DMC common command to enter the new macro. The second option is to use the *PMC command. This command will purge *all* macros from the system, and thus will require a new download of macros. This is usually not used to redefine one macro, but rather all macros.

The command sequence for a macro may contain passed parameters. The PMC200-P supports up to nine parameters in a single command line, in accordance with the IEEE-488.2 specification. In the command line, each parameter is represented by **$**<*digit*>, where <*digit*> is a single number from 1 through 9. These parameters may be used more than once, may be in any order, and may be of any type.

The contents of any macro may be retrieved by use of the *GMC? common command. A list of all macros defined may be retrieved by use of the *LMC? common command.

Returns: NONE

Examples:

Send: *DMC GOHOME,”MOVE 0.0, 0.0"<*NL*>

Defines the command macro GOHOME whose function is to move the actuators to position 0.0,0.0

Send: *DMC “loop”,#0move $1,$2;*wai;home;*wai;loop<*NL*>
Send: loop 5.0 12.5<*NL*>

The macro “loop” defines two parameters which are passed to the move command. The axes will move to location 5.0, 12.5, wait until the motion is complete, return to position 0.0,0.0, wait for motion to stop, and then repeat the process. Note that this is an infinite loop, and requires manual intervention at the keyboard or a GPIB reset to halt processing. This type of macro could be used for testing when long periods of repetitious motion is required.

Related Commands: *EMC, *EMC?, *GMC?, *LMC?, *PMC, RMC

3.22 ECHO Enable/Disable Echo Mode

Syntax: ECHO <*number*>

Parameters:

<*number*> rounds to an integer in the range from -32767 to 32767. A value greater than zero enables the echo mode and a value equal to or less than zero will disable the echo mode.

Function:

This command enables or disables the echo mode for RS-232 communication.

If the echo mode is enabled the RS-232 port is interactive. That is to say, all characters sent to the PMC200-P and error messages will be echoed back over the RS-232 port. In addition the PMC200-P generates a '>' prompt for every line. As the user is entering commands the line may be edited by using the backspace key (sending an ASCII decimal 08 code) or by using the DEL key (sending an ASCII decimal 127 code). This mode is useful when the PMC200-P is inferfacing with a dumb terminal type of device.

If the echo mode is disabled the RS-232 port is not interactive. The characters sent to the PMC200-P and error messages are not returned immediately. The error messages can be accessed by sending a *ERR? command to the PMC200-P followed by reading the string output from the PMC200-P.

Returns: NONE

Example:

This examples enables the echo mode and shows that the '>' prompt will be sent by the PMC200-P.

Send:	ECHO 1<*NL*>
Send:	<*NL*>
Receive:	>

Related Commands: ECHO?

Syntax: ECHO?

3.23 ECHO? Query Echo Mode

Parameters: NONE

Function:

This query returns a value showing whether the echo mode is enabled or disabled.

If the echo mode is enabled the RS-232 port is interactive. That is to say, all characters sent to the PMC200-P and error messages will be echoed back over the RS-232 port. In addition the PMC200-P generates a '>' prompt for every line. As the user is entering commands the line may be edited by using the backspace key (sending an ASCII decimal 08 code) or by using the DEL key (sending an ASCII decimal 127 code). This mode is useful when the PMC200-P is inferfacing with a dumb terminal type of device.

If the echo mode is disabled the RS-232 port is not interactive. The characters sent to the PMC200-P and error messages are not returned immediately. The error messages can be accessed by sending a *ERR? command to the PMC200-P followed by reading the string output from the PMC200-P.

Returns: <*number*><*NL*>

<*number*> is an unsigned integer with the value of zero if the echo mode is disabled, or a value of one if the echo mode is enabled.

Example:

```
Send:      ECHO 0<NL>
Send:      ECHO?<NL>
Receive:   0<NL>
```

Related Commands: ECHO

3.24 *EMC Enable Macro Contents

Syntax: *EMC <*number*>

Parameters:

<*number*> when rounded is in the range from -32767 to 32767. A non-zero value enables the expansion (interpretation) of macros, while a zero value disables the expansion of macros.

Function:

This command enables and disables expansion (interpretation) of macros. The macro definitions are not affected by this command. The factory default is for macros to be disabled on power-up. The *RST common command also disables the expansion of macros. The current enabled or disabled state may be retrieved by using the *EMC? common command.

When macro expansion is enabled the PMC200-P will first try to expand, or interpret, a command string as a macro name defined by the *DMC command. However, if no user defined macro name matches the command string, the PMC200-P will execute the command string as a standard command (i.e. one of the commands described in this manual).

When macro expansion is disabled the PMC200-P will execute the given command string only as a standard command and not as a user defined macro name.

Returns: NONE

Examples:

This command sequence will redefine the MOVE command to accept only one parameter.

```
Send:    *DMC "MOVE","*EMC 0;MOVE $1,2.0; *EMC 1"<NL>
Send:    MOVE 10.2<NL>
```

This will move axis 1 to the user specified position, while axis 2 will always move to 2.0. The *EMC 0 command is necessary to allow the MOVE command to be the device specific command, and not the macro. After the move is started, *EMC 1 is executed to enable macro expansion, thereby enabling the macro MOVE command. Note that the macro name is given as a quoted string.

Related Commands: *DMC, *EMC, *EMC?, *GMC?, *LMC?, *PMC, RMC

3.25 *EMC? Enable Macro Contents Query

Syntax: *EMC?

Parameters: NONE

Function:

This query returns a value showing whether macro expansion (interpretation) is enabled or disabled. Macros are enabled by using the *EMC command with a non-zero parameter. Macros are disabled by using the *EMC command with a zero parameter.

When macro expansion is enabled the PMC200-P will first try to expand, or interpret, a command string as a macro name defined by the *DMC command. However, if no user defined macro name matches the command string, the PMC200-P will execute the command string as a standard command (i.e. one of the commands described in this manual).

When macro expansion is disabled the PMC200-P will execute the given command string only as a standard command and not as a user defined macro name.

Returns: <*number*><*NL*>

<*number*> is an unsigned integer with the value of zero if macro expansion is disabled, or a value of one if macro expansion is enabled.

Examples:

```
Send:      *EMC -20<NL>
Send:      *EMC?<NL>
Receive:   1<NL>
Send:      *EMC 0.0<NL>
Send:      *EMC?<NL>
Receive:   0<NL>
```

Related Commands: *DMC, *EMC, *GMC?, *LMC?, *PMC, RMC

3.26 *ERR? Error Message Query

Syntax: *ERR?

Parameters: NONE

Function:

Returns the first (most recent) error code found in the error queue for the specific IO device (RS-232 or GPIB). The error code is also returned with a text description of the error. A maximum of 20 errors can be stored in the queue for any device. Errors that occur due to input from RS-232 are not stored in the GPIB queue, and vice versa. If no errors are stored in the queue, the PMC200-P responds with a message to that effect.

The GPIB device will always have an error number and description queued when any of the error event bits is set in the status reporting system for the GPIB device. RS-232 only uses the queue when in non-echo mode; when in echo mode, the error numbers and descriptions are automatically given to the user when they occur.

The GPIB command *CLS will clear the error queues.

Returns: *<errno> <description><NL>*

<errno> is a negative integer, in the range of -100 to -499, which is defined by the PMC200-P. *<description>* is of type *<string>*, and describes the error. For more information on error types and descriptions, see Appendix 4.3.

Examples:

```
Send:      ESE #B00111000<NL>
Send:      *ESE?<NL>
Receive:   24<NL>
Send:      *ERR?<NL>
Receive:   -100, Syntax error
```

The return value on the *ESE? command does not reflect the desired value, so the user queries for errors. The syntax error was due to the user not preceding the ESE command with an asterisk.

```
Send:      *ERR?<NL>
Receive:   0, No errors<NL>
```

On this error query, there were no errors queued up, so the PMC200-P will display this message.

Related Commands: *CLS, *STB?, EVENT?, *IST?, *STB?

3.27 *ESE Event Status Enable

Syntax: *ESE<*number*>

Parameters:

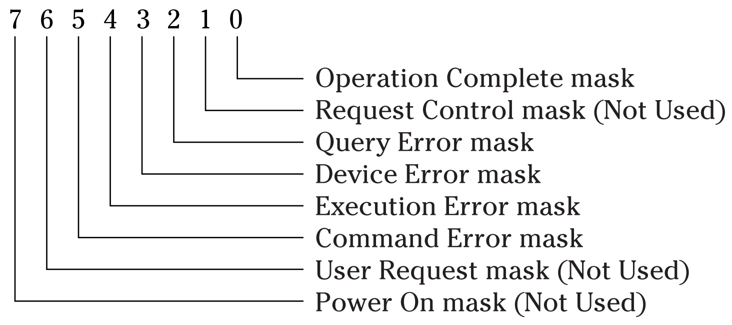
<*number*> rounds to an integer in the range from 0 to 255. This value is the Standard Event Enable Register which is the mask used with the Standard Event Register to generate the ESB bit (bit 5) of the Status Byte. The power-up default is 253.

Function:

The Event Status Enable Register is AND'ed with the Event Status Register, producing the ESB bit (bit 5) of the Status Byte. The Status Byte is used in conjunction with the Service Request Enable Register for the generation of the serial or parallel poll.

Any bit set to one in the Standard Event Enable Register enables the corresponding condition bit in the Standard Event Register to set the ESB bit (bit 5) of the Status Byte. Any bit set to zero disables the corresponding condition bit in the Standard Event Register from setting the ESB bit (bit 5) of the Status Byte.

The Standard Event Enable Register is bit mapped, with each bit signifying a different condition. The bits are listed below, most significant bit first:



Returns: NONE

Examples:

Send:	*ESE #B00111000<*NL*>
Send:	*ESE?<*NL*>
Receive:	56<*NL*>

Either a Command Error, a Execution Error, or a Device Error will cause the ESB bit in the Status Byte to be set to a one. The query form always returns the value of the Standard Event Enable Register in decimal format.

Send: *CLS; *ESE 1; MOVE1 10; *OPC<*NL*>

First all status bits are cleared. The Standard Event Enable Register is set allowing the Operation Complete bit going true to enable the ESB bit (bit 5) of the status register. Axis 1 is requested to move to absolute location 10.00. Once the motion is complete, the *OPC command enables the setting of the Operation Complete bit in the Standard Event Register. By enabling the ESB mask bit in the Service Request Enable Register, a serial or parallel poll could be requested upon completion of the motion.

Related Commands: *CLS, *ESE?, *ESR, EVENT, *IST?, *RST, *SRE, *STB?

3.28 *ESE? Event Status Enable Query

Syntax: *ESE?

Parameters: NONE

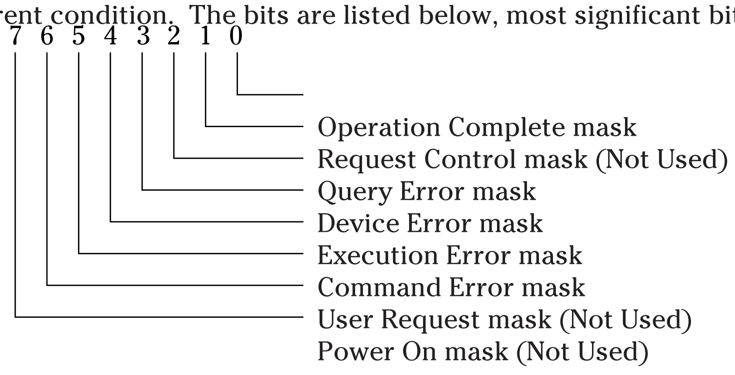
Function:

Returns the current value of the Standard Event Enable Register. The Standard Event Enable Register is AND'ed with the Standard Event Register, producing the ESB bit (bit 5) of the Status Byte. The Status Byte is used in conjunction with the Service Request Enable Register for the generation of the serial or parallel poll.

Any bit set to one in the Standard Event Enable Register enables the corresponding condition in the Standard Event Register to set the ESB bit (bit 5) of the Status Byte. Any bit set to zero disables the corresponding condition in the Standard Event Register from setting the ESB bit (bit 5) of the Status Byte.

The Standard Event Enable Register is preset at power-up.

The Standard Event Enable Register is bit mapped, with each bit signifying a different condition. The bits are listed below, most significant bit first:



Returns: *<number><NL>*

<number> is an unsigned integer in the range from 0 to 255, representing the value of the Standard Event Enable Register.

Examples:

Send:	*ESE #B00111000<NL>
Send:	*ESE?<NL>
Receive:	56<NL>

Either a Command Error, a Execution Error, or a Device Error will cause the ESB bit in the Status Byte to be set to a one. The query form always returns the value of the Standard Event Enable Register in decimal format.

Related Commands: *CLS, *ESE?, *ESR, EVENT, *IST?, *RST, *SRE, *STB?

3.29 *ESR? Event Status Register Query

Syntax: *ESR?

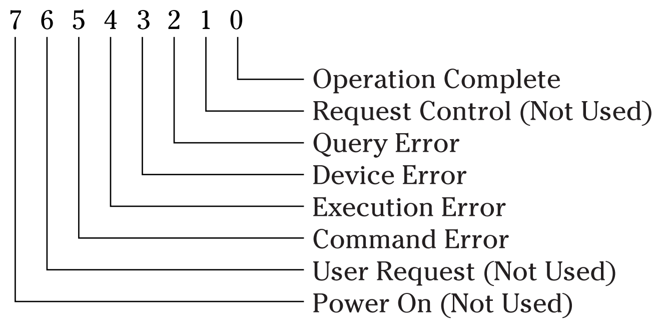
Parameters: NONE

Function:

This query returns the value stored in the Standard Event Status Register. After returning the current value, the Standard Event Status Register is cleared to zero. The *CLS common command also clears the Standard Event Status Register to zero.

The Standard Event Enable Register is AND'ed with the Standard Event Register, producing the ESB bit (bit 5) of the Status Byte. The Status Byte is used in conjunction with the Service Request Enable Register for the generation of the serial or parallel poll.

The Standard Event Status Register is bit mapped, with each bit signifying a different condition. The bits are listed below, most significant bit first:



Returns: *<number><NL>*

<number> is an unsigned integer in the range from 0 to 255 representing the value of the Standard Event Status Register. The query always returns the value in decimal format.

Examples:

Send:	*ESR?*<NL>*
Receive:	36 *<NL>*
Send:	*ESR?*<NL>*
Receive:	0*<NL>*

The Command Error bit and the Query Error bit are set. The Second query returned zero because the first query automatically cleared the Standard Event Status Register.

Related Commands: *CLS, *ESE?, *ESR, EVENT, *IST?, *RST, *SRE

3.30 *EVENT? Device Event Register Query

Syntax: EVENT?

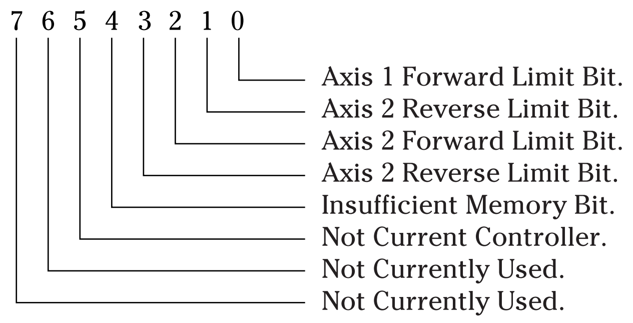
Parameters: NONE

Function:

The Device Event Register records device error events as they occur. The Device Event Register stores error codes generated by errors such as not current controller, insufficient memory, axis limits, and stalled motors. Each event is bit mapped. The Device Event Enable Register is AND'ed with the Device Event Register, producing the Device Error Bit (bit 3) in the Standard Event Status Register. Using the *ESE common command along with the *SRE common command or the *PRE common command, it is possible to request a serial poll or parallel poll from the controller.

The bits in the Device Event Register are sticky: once they are set, they remain set until the Device Event Register is queried by the EVENT? command or cleared by the *CLS command. The Device Event Register is a bit mapped value, with each bit representing a different condition.

The bits are listed below, most significant bit first:


7 6 5 4 3 2 1 0
Axis 1 Forward Limit Bit.
Axis 2 Reverse Limit Bit.
Axis 2 Forward Limit Bit.
Axis 2 Reverse Limit Bit.
Insufficient Memory Bit.
Not Current Controller.
Not Currently Used.
Not Currently Used.

Returns: *<number>*

<number> is an unsigned integer in the range from 0 to 255 representing the Device Event Register. The query always returns the value in decimal format. The Device Event Register is a bit mapped value, with each bit representing a different condition. After reading the Device Event Register, the Device Event Register is cleared.

Examples:

Send:	*CLS*<NL>*
Send:	EVENT?*<NL>*
Receive:	0*<NL>*

The *CLS command clears the Device Event Register.

Send:	RUN1 40*<NL>*

Wait until the axis 1 axis hits the positive limit.

Send:	EVENT?*<NL>*
Receive:	1*<NL>*
Send:	EVENT?*<NL>*
Receive:	0*<NL>*

When the axis 1 axis reaches a positive limit, bit 0 of the Device Event Register is set. Reading the Device Event Register will then return the value one in binary format. Reading the Device Register again will return the value of zero, since the first reading cleared the register.

Related Commands: *ESE, *ESR?, EVENTEN, EVENTEN?,

3.31 EVENTEN Device Event Enable Register

Syntax: EVENTEN <*number*>

Parameters:

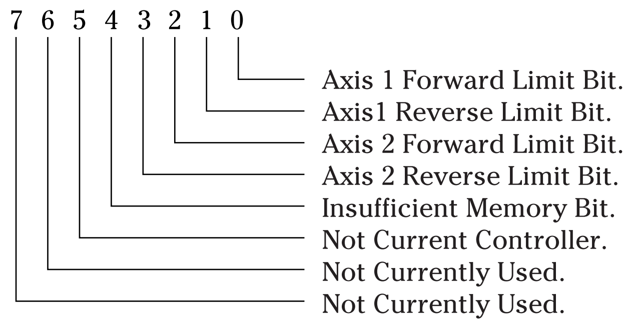
<*number*> rounds to an unsigned integer in the range from 0 to 255 representing the Device Event Enable Register. The Device Event Enable Register is a bit mapped value, with each bit representing a different mask condition. The power-up default is 0.

Function:

The Device Event Enable Register determines what events in the Device Event Register will set the Device Error bit (bit 3) in the Standard Event Status Register.

Each bit in the Device Event Enable Register is a mask bit for the corresponding condition in the Device Event Register. If any bit in the Device Event Register is set and its corresponding bit is set in the Device Event Enable Register, then the Device Error bit (bit 3) in the Standard Event Enable Register will be set. By use of the *ESE common command along with the *SRE common command or the *PRE common command, a <SRQ> could be generated by any conditions in the Device Event Register.

The Device Event Enable Register is bit mapped; each bit is listed below, most significant bit first.



Returns: NONE

Examples:

Send:	EVENTEN #b111111<*NL*>
Send:	EVENTEN?<*NL*>
Receive:	63<*NL*>

If any limit is reached, the Device Error bit (bit 3) in the Standard Event Register would be set.

Send:	EVENTEN #b101<*NL*>
Send:	*ESE #b1000<*NL*>
Send:	*SRE #b100000<*NL*>

If any positive limit is reached, a <SRQ> will be generated. The EVENTEN command allows any positive limit to set the Device Error bit (bit 3) of the Standard Event Register. The *ESE common command enables the setting of the Device Error bit (bit 3) to set the Event Status bit (bit 5) of the Status Byte. The *SRE common command enables the setting of the Event Status bit (bit 5) to generate a GPIB Service Request <SRQ>.

Related Commands: EVENT?, EVENTEN?, *ESE, *PRE, *SRE

3.32 EVENTEN? Device Event Enable Register Query

Syntax: EVENTEN?

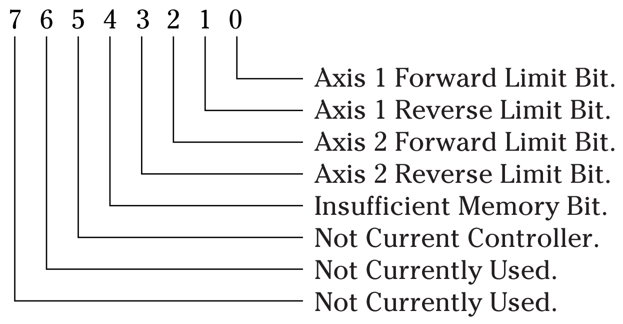
Parameters: NONE

Function:

Returns the status of the Device Event Enable Register which determines what events in the Device Event Register will set the Device Error bit (bit 3) in the Standard Event Status Register.

Each bit in the Device Event Enable Register is a mask bit for the corresponding condition in the Device Event Register. If any bit in the Device Event Register is set and its corresponding bit is set in the Device Event Enable Register, then the Device Error bit (bit 3) in the Standard Event Enable Register will be set. By use of the *ESE common command along with the *SRE common command or the *PRE common command a <SRQ> could be generated by any conditions in the Device Event Register.

The Device Event Enable Register is bit mapped, each bit is listed below, most significant bit first.


7 6 5 4 3 2 1 0
Axis 1 Forward Limit Bit.
Axis 1 Reverse Limit Bit.
Axis 2 Forward Limit Bit.
Axis 2 Reverse Limit Bit.
Insufficient Memory Bit.
Not Current Controller.
Not Currently Used.
Not Currently Used.

Returns: *<number><NL>*

<number> is an unsigned decimal integer in the range from 0 to 255 representing the Device Event Enable Register. The query always returns the value in decimal format. The Device Event Enable Register is a bit mapped value, with each bit representing a different mask condition.

Examples:

Send:	EVENTEN #b111111*<NL>*
Send:	EVENTEN?*<NL>*
Receive:	63*<NL>*

If any limit is reached, the Device Error bit (bit 3) in the Standard Event Register would be set.

Send:	EVENTEN #b101*<NL>*
Send:	*ESE #b1000*<NL>*
Send:	*SRE #b100000*<NL>*

If any positive limit is reached, a <SRQ> will be generated. The EVENTEN command allows any positive limit to set the Device Error bit (bit 3) of the Standard Event Register. The *ESE common command enables the setting of the Device Error bit (bit 3) to set the Event Status bit (bit 5) of the Status Byte. The *SRE common command enables the setting of the Event Status bit (bit 5) to generate a GPIB Service Request <SRQ>.

Related Commands:: EVENT?, EVENTEN, ESE?, PRE?, SRE?

3.33 GAINF Set Axis 1 & 2 HCTL-1100 Gain Parameter

Syntax: GAINF [<*axis 1_gainf*>][,<*axis 2_gainf*>]

Parameters:

<*axis 1_gainf*> is rounded to an integer <*number*> that represents the axis 1 gain filter parameter. The valid range for this value is 0 to 255. Values outside of this range should not be used.

<*axis 2_gainf*> is rounded to an integer <*number*> that represents the axis 2 gain filter parameter. The valid range for this value is 0 to 255. Values outside of this range should not be used.

Function:

This command sets a servo filter parameter used by the HCTL-1100 motion controller chip on the PMC200-P. This parameter effects actuator motion during a HOME, MOVE, JOG, and RUN command. The gain filter is the only filter parameter used during a RUN so it will have the biggest effect on this type of motion.

Reference:

For a more technical explanation of the gain filter parameter consult a copy of the Hewlett Packard Application Note 1032.

Example:

Send: GAINF ,100<*NL*>

Sets the gain filter parameter of axis 2 to a value of 100.

Related Commands: GAINF?, POLEF, POLEF?, ZEROF, and ZEROF?

3.34 GAINF? Axis 1 & 2 HCTL-1100 Gain Parameter Query

Syntax: GAINF?

Parameters: NONE

Function:

This command returns the servo filter parameters used by the HCTL-1100 motion controller chips on the PMC200P. This parameter effects actuator motion during a HOME, MOVE, JOG, and RUN command. The gain filter is the only filter parameter used during a RUN so it will have the biggest effect on this type of motion.

Problems with system stability can occur when the gain filter is not set correctly, especially on rotary stages. A gain filter set too high may cause jitter in the actuators during a motion. If the gain filter is too low then the average velocity of the actuator during a RUN may be lower then the velocity requested.

Reference:

For a more technical explanation of the gain filter parameter consult a copy of the Hewlett Packard Application Note 1032.

Returns: *<axis 1_gainf>,<axis 2_gainf><NL>*

<axis 1_gainf>,*<axis 2_gainf>* are of type *<number>* and represent the current settings of the gain filter parameter for axes 1 and 2.

Example:

Send:	GAINF?*<NL>*
Receive:	200,200*<NL>*

Queries the current gain filter parameter for axes 1 and 2.

Related Commands: GAINF, POLEF, POLEF?, ZEROF, and ZEROF?

3.35 *GMC? Get Macro Contents Query

Syntax: *GMC? <*name*>

Parameters:

<*name*> is of type <*string*> and is the name of the macro to retrieve the contents of. There is no case conversion on macro names, and names must conform to the standard macro name specification. The length of the name is not restricted, but only the first 16 characters are significant.

Function:

This command retrieves the contents of the requested macro. If <*name*> is an unknown macro name, then an error is generated.

Returns: <*string*><*NL*>

<*string*> is the contents of the requested macro returned using the block length format. If string is a null string, then a zero length block is returned.

Examples:

Send:	*DMC “STOPXY”, “STOP1;STOP2”<*NL*>
Send:	*GMC? STOPXY<*NL*>
Receive:	#211STOP1;STOP2<*NL*>

Related Commands: *DMC, *EMC, *EMC?, *LMC?, *PMC, RMC

3.36 HOME Move Axis 1 & Axis 2 to Origin

Syntax: HOME

Parameters: NONE

Function:

The HOME command moves the axis 1 and axis 2 actuators to the zero position. The motion is started from the current position, and terminates when the actuators have all been positioned at the currently defined zero position.

On power-up condition, the zero position is defined to be the actuators' position at power-up. The ZERO command allows the user to redefine this value at any time after power up.

The actuators are moved to the zero position by using a trapezoidal motion trajectory, computed by the HP HCTL-1100 motion control processor. A move is computed by accelerating until the maximum velocity is reached or until deceleration must begin to stop at the specified final position. The deceleration rate is equal to the acceleration rate. Maximum velocity for the move can be modified using the VEL commands.

The HOME command is equivalent to executing the HOME1 and HOME2 commands simultaneously.

While the HOME commands could be simulated by the MOVE 0,0 command, the HOME command will not affect the current move destination values stored by the MOVE, MOVE1, or MOVE2 commands.

Returns: NONE

Examples:

Send: HOME<*NL*>

Will move Axis 1 and Axis 2 actuators to their currently defined ZERO position.

Related Commands: HOME1, HOME2, ZERO, ACL, MOVE

3.37 HOME1 Move Axis 1 to Origin

Syntax: HOME1

Parameters: NONE

Function:

The HOME command moves the axis 1 actuator to the zero position. The motion is started from the current position, and terminates when the actuator has been positioned at the currently defined zero position.

On power-up condition, the zero position is defined to be the actuator's position at power-up. The ZERO command allows the user to redefine this value at any time after power up.

The actuator is moved to the zero position by using a trapezoidal motion trajectory, computed by the HP HCTL-1100 motion control processor. A move is computed by accelerating until the maximum velocity is reached or until deceleration must begin to stop at the specified final position. The deceleration rate is equal to the acceleration rate. Maximum velocity for the move can be modified by using the VEL commands.

While the HOME1 command could be simulated by the MOVE1 0 command, the HOME1 command will not affect the current move destination values stored by the MOVE or MOVE1 commands.

Returns: NONE

Examples:

Send: HOME1<NL>

Will move the Axis 1 actuator to the currently defined 0 position.

Related Commands: HOME, HOME2, ZERO1, ACL1, MOVE1

3.38 HOME2 Move Axis 2 to Origin

Syntax: HOME2

Parameters: NONE

Function:

The HOME command moves the axis 2 actuator to the zero position. The motion is started from the current position, and terminates when the actuator has been positioned at the currently defined zero position.

On power-up condition, the zero position is defined to be the actuator's position at power-up. The ZERO command allows the user to redefine this value at any time after power up.

The actuator is moved to the zero position by using a trapezoidal motion trajectory, computed by the HP HCTL-1100 motion control processor. A move is computed by accelerating until the maximum velocity is reached or until deceleration must begin to stop at the specified final position. The deceleration rate is equal to the acceleration rate. Maximum velocity can be modified by using the VEL commands.

While the HOME2 command could be simulated by the MOVE2 0 command, the HOME2 command will not affect the current move destination values stored by the MOVE or MOVE2 commands.

Returns: NONE

Examples:

Send: HOME2<*NL*>

Will move the Axis 2 actuator to the currently defined 0 position.

Related Commands: HOME, HOME1, ZERO2, ACL2, MOVE2

3.39 *IDN? Identification Query

Syntax: *IDN?

Parameters: NONE

Function:

This query causes the PMC200-P to return identification information. The information returned is manufacturer, model, serial number, and firmware version.

Returns: *<manufacturer>,<model>,<serial>,<version><NL>*

<*manufacturer*> is of type <*string*> using the no quotes format. The PMC200-P always returns Newport Corp.

<*model*> is of type <*string*> using the no quotes format. The PMC200-P always returns PMC200-P.

<*serial*> is of type <*string*> using the no quotes format. This is always a ‘0’, as the serial number is not actually returned.

<*version*> is of type <*string*> using the no quotes format. This text identifies the internal firmware version of the PMC200-P. This text consists of two sections, separated by two underscores. The first is the revision level, and the second is revision date.

Examples:

Send:	*IDN?<*NL*>
Receive:	Newport Corp,PMC200-P,0,1.0_060189<*NL*>

Related Commands: NONE

3.40 *IST? Individual Status Query (GPIB only)

Syntax: *IST?

Parameters: NONE

Function:

This query returns the current state of the IEEE 488.1 <IST> local message. The <IST> message is generated by the parallel poll status system. The *PRE common command is used to defined the Parallel Poll Enable Register used with the Status Byte to generate the <IST> message. If any bit is set in the Status Byte and its corresponding bit is set in the Parallel Poll Enable Register, then the <IST> message is set true (a value of one). Otherwise the <IST> message is set false (a value of zero).

The <IST> message is compared to the <S BIT> register, and if they are the same and the Parallel Poll Register has a non-zero value in it, then a Parallel Poll is requested and the Parallel Poll Register will be returned when the PMC200-P has been polled in a parallel poll.

Both the <S BIT> and the Parallel Poll Register may be configured by the IEEE 488.1 Parallel Poll remote configuration command.

Returns: *<number><NL>*

<number> is an unsigned integer in the range from 0 to 1. A value of zero indicates that no bit is set in the Status Byte it's corresponding bit is set in the Parallel Poll Enable Register. A value of one indicates that a bit is set in the Status Byte and its corresponding bit is set in the Parallel Poll Enable Register.

Examples:

```
Send:      *CLS; *IST?<NL>
Receive:   0<NL>
```

Using the *CLS command clears all status bytes, including the IST message.

```
Send:      *CLS; *ESE #B100000; *PRE #B100000<NL>
Send:      this_is_garbage!!!<NL>
Send:      *IST?<NL>
Receive:    1<NL>
```

The Standard Event Enable Register is set to allow a command error to set the ESB bit (bit 5) in the Status Byte. The Parallel Poll Enable Register is set to enable the ESB bit (bit 5) of the Status Byte to set the <IST> message. Sending this_is_garbage!!! will cause a command error and subsequently set the <IST> message. If the Parallel Poll Register has been set-up using remote configuration, then a parallel poll would be requested.

Related Commands: *PRE, *PRE?

3.41 JOG Move Axis 1 & 2 to Relative Destination

Syntax: JOG [<*Axis 1_arg*>][,<*Axis 2_arg*>]

Parameters:

<*Axis 1_arg*> converts to a floating point <*number*> representing the relative distance, in currently defined units, to move the axis 1 actuator. If the <*Axis 1_arg*> parameter is missing, the last relative displacement defined by the JOG or JOG1 commands is used. The factory default value is 0.0 units.

<*Axis 2_arg*> converts to a floating point <*number*> representing the relative distance, in currently defined units, to move the axis 2 actuator. If the <*Axis 2_arg*> parameter is missing, the last relative displacement defined by the JOG or JOG2 commands is used. The factory default value is 0.0 units.

Function:

This command causes the actuators to move a specified number of units. Both Axis 1 and Axis 2 are affected by the command. The displacement is measured relative to the present position.

During motion, the actuator is accelerated up to the maximum acceleration until either the maximum velocity is reached or the point where the actuator must be de-accelerated to reach the target position.

The JOG command is functionally equivalent to executing the JOG1 and JOG2 commands simultaneously.

Returns: NONE

Examples:

```
Send:      ZERO<NL>
Send:      JOG 30.5, -100.0<NL>
Send:      JOG?<NL>
Receive:   30.50,-100.00<NL>
Send:      *WAI; POS?
Receive:   30.50,-100.00<NL>
```

Will move the Axis 1 axis 30.5 units forward, and the Axis 2 axis 100.0 units backwards.

```
Send:      JOG ,-10.54<NL>
Send:      JOG?<NL>
Receive:   30.50,-10.54<NL>
Send:      *WAI; POS?<NL>
Receive:   61.00,-110.54<NL>
```

Will move the Axis 1 axis forward an additional 30.5 units, and the Axis 2 axis backwards an additional 10.54 units

Related Commands: JOG?, JOG1, JOG2, MOVE, MOVE1, MOVE2, VEL, VEL1, VEL2, ACL

3.42 JOG? Axis 1 & 2 to Relative Destination Query

Syntax: JOG?

Parameters: NONE

Function:

This command returns the displacement values currently defined for axis 1 and axis 2 relative motion.

The relative displacement values may be defined by the JOG, JOG1, or JOG2 commands.

Returns: <*Axis 1_arg*>,<*Axis 2_arg*><*NL*>

<*Axis 1_arg*> is a floating point <*number*> representing the relative distance, in currently defined units, to move the axis 1 actuator. The returned value is in standard floating point form, with varying levels of precision, based on current units defined for each actuator.

<*Axis 2_arg*> is a floating point <*number*> representing the relative distance, in currently defined units, to move the axis 2 actuator. The returned value is in standard floating point form, with varying levels of precision, based on current units defined for each actuator.

Examples:

Send:	JOG 1.1, 2.2<*NL*>
Send:	JOG?<*NL*>
Receive:	001.100,002.200<*NL*>

Related Commands: JOG, JOG1, JOG1?, JOG2, JOG2?

3.43 JOG1 Move Axis 1 to Relative Destination

Syntax: JOG1 [<*Axis 1_arg*>]

Parameters:

<*Axis 1_arg*> converts to a floating point <*number*> representing the relative distance, in currently defined units, to move the axis 1 actuator. If the <*Axis 1_arg*> parameter is missing, the last relative displacement defined by the JOG or JOG1 commands is used. The factory default value is 0.0 units.

Function:

This command causes the axis 1 actuator to move a specified number of units. The displacement is measured relative to the present position.

During motion, the actuator is accelerated up to the maximum acceleration until either the maximum velocity is reached or the point where the actuator must be de-accelerated to reach the target position.

The JOG1 command is used when it is required that the axis 2 actuator not be moved simultaneously with the axis 1 actuator, as when using the JOG command.

Returns: NONE

Examples:

```
Send:      ZERO1<NL>
Send:      JOG1 30.5<NL>
Send:      *WAI; POS1?<NL>
Receive:   030.500<NL>
```

Will move the axis 1 actuator 30.5 units forward.

```
Send:      JOG1<NL>
Send:      *WAI;POS1?<NL>
Receive:   061.000<NL>
```

Will move axis 1 forward an additional 30.5 units.

Related Commands: JOG, JOG?, JOG1?, JOG2, JOG2?, VEL, VEL1, ACL

3.44 JOG1? Axis 1 Relative Destination Query

Syntax: JOG1?

Parameters: NONE

Function:

This command returns the relative displacement value, in currently defined units, that will be used to by the JOG command or the JOG1 command to position the axis 1 actuator.

The relative displacement value may be defined by the JOG or JOG1 commands.

Returns: *<Axis 1_arg><NL>*

<Axis 1_arg> is a floating point *<number>* representing the relative distance to move the axis 1 actuator. The returned value is of standard floating point form, with varying levels of precision based on the currently defined units. The factory default value is 0.00 units.

Examples:

```
Send:      ZERO1<NL>
Send:      JOG1 30.5<NL>
Send:      JOG1?<NL>
Receive:   030.500<NL>
```

Returns the defined relative motion displacement for axis 1.

```
Send:      JOG1<NL>
Send:      *WAI;POS1?<NL>
Receive:   061.000<NL>
Send:      JOG1?<NL>
Receive:   030.500<NL>
```

Note that the displacement is the same, and no parameter was given with the JOG1 command.

Related Commands: JOG, JOG?, JOG1, JOG2, JOG2?,

3.45 JOG2 Move Axis 2 to Relative Destination

Syntax: JOG2 [<*Axis 2_arg*>]

Parameters:

<*Axis 2_arg*> converts to a floating point <*number*> representing the relative distance, in currently defined units, to move the axis 2 actuator. If the <*Axis 2_arg*> parameter is missing, the last relative displacement defined by the JOG or JOG2 commands is used. The factory default value is 0.0 units.

Function:

This command causes the axis 2 actuator to move a specified number of units. The displacement is measured relative to the present position.

During motion, the actuator is accelerated up to the maximum acceleration until either the maximum velocity is reached or the point where the actuator must be de-accelerated to reach the target position.

The JOG2 command is used when it is required that the axis 1 actuator not be moved simultaneously with the axis 2 actuator, as when using the JOG command.

Returns: NONE

Examples:

Send:	ZERO2<*NL*>
Send:	JOG2 30.5<*NL*>
Send:	*WAI; POS2?<*NL*>
Receive:	030.500<*NL*>

Will move the axis 2 actuator 30.5 units forward.

Send:	JOG2<*NL*>
Send:	*WAI;POS2?<*NL*>
Receive:	061.000<*NL*>

Will move axis 2 forward an additional 30.5 units.

Related Commands: JOG, JOG?, JOG1, JOG1?, JOG2?, VEL, VEL2, ACL

3.46 JOG2? Axis 2 Relative Destination Query

Syntax: JOG2?

Parameters: NONE

Function:

This command returns the relative displacement value, in currently defined units, that will be used to by the JOG command or the JOG2 command to position the axis 2 actuator.

The relative displacement value may be defined by the JOG or JOG2 commands.

Returns: *<Axis 2_arg><NL>*

<Axis 2_arg> is a floating point *<number>* representing the relative distance to move the axis 2 actuator. The returned value is of standard floating point form, with varying levels of precision based on the currently defined units. The factory default value is 0.00 units.

Examples:

```
Send:       ZERO2<NL>
Send:       JOG2 30.5<NL>
Send:       JOG2?<NL>
Receive:    030.500<NL>
```

Returns the defined relative motion displacement for axis 2.

```
Send:       JOG2<NL>
Send:       *WAI;POS2?<NL>
Receive:    061.000<NL>
Send:       JOG2?<NL>
Receive:    030.500<NL>
```

Note that the displacement is the same, and no parameter was given with the JOG2 command.

Related Commands: JOG, JOG?, JOG1, JOG1?, JOG2

3.47 *LMC? Learn Macro Contents Query

Syntax: *LMC?

Parameters: NONE

Function:

This query returns all of the currently defined macro labels. The macro names are returned in the order they were defined. If no macros have been defined, then a null string “” is returned. This command does not return the contents of macros, only a directory of the macro names defined.

Returns: *<name>*,*<name>*,*<name>*, ... ,*<name><NL>*

<name> is of type *<string>*, using the double quotes format. Each *<name>* is separated from the next *<name>* by a comma. The last *<name>* is terminated with a *<NL>*. If no macros are currently defined, then a null string “” is returned.

Examples:

```
Send:       *DMC “STOPXY”, “STOP1;STOP2”<NL>
Send:       *DMC HOMEXY, ‘HOME1; HOME2’<NL>
Send:       *DMC test,#0MOVE1 10;*WAI;HOME<NL>
Send:       *LMC?<NL>
Receive:    “STOPXY”,“HOMEXY”,“test”<NL>
```

Each name is enclosed in double quotes and separated by a comma. Notice that the macro contents are not returned. Note also that case conversion is not performed on the macro names.

```
Send:       *PMC<NL>
Send:       *LMC?<NL>
Receive:    “”<NL>
```

The *PMC common command removes all defined macros. Using the *LMC common command afterward returns a null string since no macros are defined.

Related Commands: *DMC, *EMC, *EMC?, *GMC?, *PMC, RMC

3.48 MOTION Define Actuator Motion Type

Syntax: MOTION [<*axis 1_motion*>][,<*axis 2_motion*>]

Parameters:

<*axis 1_motion*> is of type <*string*>, and is used to set the motion type for the actuator attached to axis 1.

<*axis 2_motion*> is of type <*string*>, and is used to set the motion type for the actuator attached to axis 2.

Function:

This command defines the motion type for the actuators, if any, that are currently attached to the PMC200-P on axis 1 and axis 2. The default motion type is dependent on which actuator is selected. The defined motion types are LINEAR and ROTARY. The table below gives the supported actuators and their default motion types.

Newport 495 actuator	Rotary
Newport 496 actuator	Rotary
Newport 850, 850A or 850B actuator	Linear
Non-standard actuator	Linear
No actuator attached	Linear

The MOTION command is not case-sensitive with respect to parameters. For example, “ROTARY” is the same as “rotary”. Only the first character in a parameter string is significant.

The motion type also determines the units in which all positional and velocity values are measured. Linear motion forces the use of either millimeters or inches. Rotary motion forces the use of degrees or milliradians. If the PMC200-P is using units that don’t correspond to the actual motion for an actuator, erroneous data will be generated. The SCALE command can be used in this case to re-scale the data.

Returns: NONE

Examples:

```
Send:      ACTTYP “495”,“850”<NL>
Send:      MOTION?<NL>
Receive:   “rotary”,“linear”<NL>
Send:      MOTION “linear”<NL>
Send:      MOTION?<NL>
Receive:   “linear”,“linear”<NL>
```

The above example sets all axis 1 operational parameters for a 495-type rotary actuator, and axis 2 for an 850-type actuator. The MOTION command is then used to force axis 1 to use linear motion.

Related Commands: ACTTYP, ACTTYP?, MOTION?, MTRTYP, MTRTYP?, CPLRAT, CPLRAT?

3.49 MOTION? Actuator Motion Type Query

Syntax: MOTION?

Parameters: NONE

Function:

This query returns the defined motion type for the actuators, if any, that are currently attached to the PMC200-P on axis 1 and axis 2. The default for each axis is determined by the actuator selection. Supported actuator types and their default motion type is given below.

Newport 495 actuator	Rotary
Newport 496 actuator	Rotary
Newport 850 or 850A actuator	Linear
Non-standard actuator	Linear
No actuator attached	Linear

The motion type also determines the units in which all positional and velocity values are measured. Linear motion forces the use of either millimeters or inches. Rotary motion forces the use of degrees or milliradians. If the PMC200-P is using units that don't correspond to the actual motion for an actuator, erroneous data will be generated. The SCALE command can be used in this case to re-scale the data.

Returns: <*axis 1_motion*>,<*axis 2_motion*>

<*axis 1_motion*> is of type <*string*>, and represents the motion type for the actuator attached to axis 1. Motion type will only be linear or rotary.

<*axis 2_motion*> is of type <*string*>, and represents the motion type for the actuator attached to axis 2. Motion type will only be linear or rotary.

Examples:

Send:	ACTTYP "495","850"<*NL*>
Send:	MOTION?<*NL*>
Receive:	"rotary","linear"<*NL*>

The above example sets all axis 1 operational parameters for a 495-type rotary actuator, and axis 2 for an 850-type actuator. The MOTION query then returns the defined motion types for each axis.

Related Commands: ACTTYP, ACTTYP?, MOTION, MTRTYP, MTRTYP?, CPLRAT, CPLRAT?

3.50 MOVE Move Axis 1 & 2 to Destination

Syntax: MOVE [<*Axis 1_arg*>][,<*Axis 2_arg*>]

Parameters:

<*Axis 1_arg*> converts to a floating point <*number*> representing the absolute distance, in currently defined units, to move the axis 1 actuator. The absolute distance is measured from the zero position (origin). If the <*Axis 1_arg*> parameter is missing, the last defined absolute displacement defined by the MOVE or MOVE1 commands is used. The factory default value is 0.0 units.

<*Axis 2_arg*> converts to a floating point <*number*> representing the absolute distance, in currently defined units, to move the axis 2 actuator. The absolute distance is measured from the zero position (origin). If the <*Axis 2_arg*> parameter is missing, the last defined absolute displacement defined by the MOVE or MOVE2 commands is used. The factory default value is 0.0 units.

Function:

This command moves the actuators a specified distance, where the displacement is measured absolutely to the origins. The destinations may be defined by the MOVE, MOVE1, or MOVE2 commands.

During motion, the actuator is accelerated up to the maximum acceleration until either the maximum velocity is reached or the point where the actuator must be de-accelerated to reach the target position.

The MOVE command is functionally equivalent to the MOVE1 and MOVE2 commands being executed simultaneously.

Returns: NONE

Examples:

```
Send:       MOVE 30.5, -100.0<NL>
Send:       MOVE?<NL>
Receive:    030.500,-100.000<NL>
Send:       *WAI; POS?
Receive:    030.500,-100.000<NL>
```

Will move the axis 1 actuator to 30.5 units, and the axis 2 actuator to -100.0 units.

```
Send:       JOG 10,10;*WAI<NL>
Send:       MOVE ,10.54<NL>
Send:       MOVE?<NL>
Receive:    030.500,010.540<NL>
Send:       *WAI; POS?<NL>
Receive:    030.500,010.540<NL>
```

First use the JOG command to move the actuators 10 units from the defined move position. The MOVE command will position the axis 1 actuator back at the previously defined position and position the axis 2 actuator at 10.54 units.

Related Commands: MOVE?, MOVE1, MOVE1?, MOVE2, MOVE2?, VEL, VEL1, VEL2, ACL

3.51 MOVE? Axis 1 & 2 MOVE Destination Query

Syntax: MOVE?

Parameters: NONE

Function:

This query returns the absolute destination values that will be used to for axis 1 and axis 2 absolute motions. The absolute destination values may be defined by the MOVE, MOVE1, or MOVE2 commands.

Returns: <*axis 1_arg*>,<*axis 2_arg*><*NL*>

<*axis 1_arg*> is a floating point <*number*> representing the absolute distance, in currently defined units, to move the axis 1 actuator from the origin. The returned value is in standard floating point form, with varying levels of precision based on defined units. The factory default value is 0.00 units.

<*axis 2_arg*> is a floating point <*number*> representing the absolute distance, in currently defined units, to move the axis 2 actuator from the origin. The returned value is in standard floating point form, with varying levels of precision based on defined units. The factory default value is 0.00 units.

Examples:

Send:	MOVE 1.1, 2.2<*NL*>
Send:	MOVE? <*NL*>
Receive:	001.100,002.200<*NL*>

Related Commands: MOVE, MOVE1, MOVE1?, MOVE2, MOVE2?

3.52 MOVE1 Move Axis 1 to Destination

Syntax: MOVE1 [<*axis 1_arg*>]

Parameters:

<*Axis 1_arg*> converts to a floating point <*number*> representing the absolute distance, in currently defined units, to move the axis 1 actuator. The absolute distance is measured from the zero position (origin). If the <*Axis 1_arg*> parameter is missing, the last defined absolute displacement defined by the MOVE or MOVE1 commands is used. The factory default value is 0.0 units.

Function:

This command moves the axis 1 actuator to a specified position, where the destination is measured from the origin. The destination may be defined by MOVE or MOVE1.

During motion, the actuator is accelerated up to the maximum acceleration until either the maximum velocity is reached or the point where the actuator must be de-accelerated to reach the target position.

The MOVE1 command is used when it is required that the axis 2 actuator not be moved simultaneously with the axis 1 actuator, as when using the MOVE command.

Returns: NONE

Examples:

```
Send:       MOVE1 30.5<NL>
Send:       *WAI; POS1?<NL>
Receive:    030.500<NL>
```

Will move the axis 1 actuator to 30.5 units.

```
Send:       JOG1 10; *WAI; MOVE1<NL>
Send:       *WAI;POS1?<NL>
Receive:    030.500<NL>
```

The JOG1 command will move the axis 1 actuator 10 units from the current position. The MOVE1 command will then return the axis 1 actuator to the previously defined position.

Related Commands: MOVE, MOVE?, MOVE1?, MOVE2, MOVE2?, VEL, VEL1, ACL

3.53 MOVE1? Axis 1 MOVE Destination Query

Syntax: MOVE1?

Parameters: NONE

Function:

This command returns the axis 1 actuator absolute destination value that will be used by the MOVE command or the MOVE1 command to position the axis 1 actuator. The axis 1 actuator absolute destination value may be defined by the MOVE or MOVE1 commands.

Returns: <*axis 1_arg*><*NL*>

<*axis 1_arg*> is a floating point <*number*> representing the absolute distance, in currently defined units, to move the axis 1 actuator from the origin. The returned value is in standard floating point form, with varying levels of precision based on defined units. The factory default value is 0.00 units.

Examples:

```
Send:      MOVE1 30.5<NL>
Send:      MOVE1?<NL>
Receive:   030.500<NL>
Send:      JOG1 10; *WAI; MOVE1<NL>
Send:      *WAI;POS1?<NL>
Receive:   030.500<NL>
Send:      MOVE1?<NL>
Receive:   030.500<NL>
```

Related Commands: MOVE, MOVE?, MOVE1, MOVE2, MOVE2?

3.54 MOVE2 Move Axis 2 to Destination

Syntax: MOVE2 [<*axis 2_arg*>]

Parameters:

<*axis 2_arg*> converts to a floating point <*number*> representing the absolute distance to move the axis 2 actuator in units from the origin. If the <*axis 2_arg*> parameter is missing, the last defined axis 2 actuator absolute destination defined by the MOVE or MOVE2 commands is used. The factory default value is 0.0 units.

Function:

This command moves the axis 2 actuator to a specified position, where the destination is measured from the origin. The destination may be defined by MOVE or MOVE1.

During motion, the actuator is accelerated up to the maximum acceleration until either the maximum velocity is reached or the point where the actuator must be de-accelerated to reach the target position.

The MOVE2 command is used when it is required that the axis 1 actuator not be moved simultaneously with the axis 2 actuator, as when using the MOVE command.

Returns: NONE

Examples:

Send:	MOVE2 30.5<*NL*>
Send:	*WAI; POS2?<*NL*>
Receive:	030.500<*NL*>

Will move the axis 2 actuator to 30.5 units.

Send:	JOG2 10; *WAI; MOVE2<*NL*>
Send:	*WAI;POS2?<*NL*>
Receive:	030.500<*NL*>

The JOG2 command will move the axis 2 actuator 10 units from the current position. The MOVE2 command will then return the axis 2 actuator to the previously defined position.

Related Commands: MOVE, MOVE?, MOVE1, MOVE1?, MOVE2?, VEL, VEL2

3.55 MOVE2? Axis 2 Move Destination Query

Syntax: MOVE2?

Parameters: NONE

Function:

This command returns the axis 2 actuator absolute destination value that will be used by the MOVE command or the MOVE2 command to position the axis 2 actuator. The axis 2 actuator absolute destination value may be defined by the MOVE or MOVE2 commands.

Returns: <*axis 2_arg*><*NL*>

<*axis 2_arg*> is a floating point <*number*> representing the absolute distance, in currently defined units, to move the axis 2 actuator from the origin. The returned value is in standard floating point form, with varying levels of precision based on defined units. The factory default value is 0.00 units.

Examples:

```
Send:       MOVE2 30.5<NL>
Send:       MOVE2?<NL>
Receive:    030.500<NL>
Send:       JOG2 10; *WAI; MOVE2<NL>
Send:       *WAI;POS2?<NL>
Receive:    030.500<NL>
Send:       MOVE2?<NL>
Receive:    030.500<NL>
```

Related Commands: MOVE, MOVE?, MOVE1, MOVE1?, MOVE2

3.56 MTRTYP Define Actuator Motor Type

Syntax: MTRTYP [<*axis 1_mtr*>][,<*axis 2_mtr*>]

Parameters:

<*axis 1_mtr*> is of type <*string*>, and represents the motor type used in the actuator attached to axis 1. As the defined motor types appear to the system as integers, the string should be quoted. The currently defined values for the parameter fields are given in the table below. This command is usually used only when defining a special actuator.

<*axis 2_mtr*> is of type <*string*>, and represents the motor type used in the actuator attached to axis 2. As the defined motor types appear to the system as integers, the string should be quoted. The currently defined values for the parameter fields are given in the table below. This command is usually used only when defining a special actuator.

Defined Motor Types

1616	Linear actuator motor
1624	Linear actuator motor

Function:

This command defines the type of motor used within a linear actuator of type special, on either axis 1 or axis 2. Default motor type for linear actuator is the 1616 motor.

When a special actuator is defined, the MTRTYP and CPLRAT commands should be used to define the actuator motor type and coupling ratio (gear ratio), respectively, of the special actuator. Failure to do so can result in incorrect actuator motion and returned position and actual velocity values.

Returns: NONE

Examples:

```
Send:     ACTTYP ,"SPECIAL"<NL>
Send:     MTRTYP ,"1624"<NL>
Send:     CPLRAT ,1600<NL>
```

The above example sets axis 2 actuator type to “special”, and the MTRTYP command sets the actuator motor type to “1624”, and finally the CPLRAT command sets the gear ratio for 1600 quadrature counts per mm. The axis 1 actuator was not affected.

Related Commands: ACTTYP, ACTTYP?, MOTION, MOTION?, MTRTYP?, CPLRAT, CPLRAT?

3.57 MTRTYP? Actuator Motor Type Query

Syntax: MTRTYP?

Parameters: NONE

Function:

This query returns the defined motor types for the actuators, if any, that are currently attached to the PMC200-P on axis 1 and axis 2. The default for each axis is the 1616 motor type, if actuator type is defined as "NONE".

Returns: <*axis 1_mtr*>,<*axis 2_mtr*>

<*axis 1_act*> is of type <*string*>, and represents the motor type for the actuator attached to axis 1. The currently defined values for the parameter fields are given in the table below.

<*axis 2_act*> is of type <*string*>, and represents the motor type for the actuator attached to axis 2. The currently defined values for the parameter fields are given in the table below.

Defined Actuator Motor Types

1616	Linear actuator motor
1624	Linear actuator motor
2233	Rotary actuator motor

Examples:

Send:	ACTTYP "495","850"<*NL*>
Send:	MTRTYP?<*NL*>
Receive:	"2233","1616"<*NL*>

The above example sets all axis 1 operational parameters for a 495-type rotary actuator, and axis 2 for an 850-type actuator.

Related Commands: ACTTYP, ACTTYP?, MOTION, MOTION?, MTRTYP, CPLRAT, CPLRAT?

3.58 *OPC Operation Complete (GPIB only)

Syntax: *OPC

Parameters: NONE

Function:

This command sets the Operation Complete bit (OPC bit) of the Standard Event Status Register when all pending operations have been finished. This is to allow for synchronization of operations between the PMC200-P and the remote host. This command is only supported for the GPIB port.

Bit 0 of the Standard Event Status Register is the OPC bit. Normally this bit is set to zero. The *OPC common command forces the PMC200-P to set the OPC bit when all pending operations have been finished.

A pending operation is any command that causes the actuators to move. HOME, JOG, MOVE, and RUN commands all move the actuators. so the *OPC common command could be used to set bit 0 of the Standard Event Status Register when they have completed their motion. The STOP command will terminate all pending operations.

By enabling bit 0 of the Standard Event Enable Register, the *OPC common command allows for the ESB bit (bit 5) of the Status Byte to be set when all pending operations are finished.

The *OPC common command and the *OPC? query operate differently in how they signal an operation complete to the remote host. The *OPC? query generates a message ("1<*NL*>") when all operations are complete, which also sets the MAV bit (bit 4) in the status byte.

Returns: NONE

Examples:

Send:	*CLS; *ESE 1; *SRE #B00100000<*NL*>
Send:	MOVE; *OPC<*NL*>

When the MOVE command has finished executing, bit 0 of the Standard Event Status Register will be set. Since bit 0 of the Standard Event Enable Register is also set, bit 5 of the Status byte will be set. Since bit 5 of the Service Request Enable Register is set, a GPIB <SRQ> will be generated.

Related Commands: OPC?, *OPC?, *WAI, *ESR?, *ESE

3.59 *OPC? Operation Complete

Syntax: *OPC?

Parameters: NONE

Function:

Generates a response when all pending operations have been completed. The Message Available bit (bit 4) of the Status Byte will also be set because a response is generated. This query is only supported on the GPIB port.

A pending operation is any command that causes the actuators to move. HOME, JOG, MOV, and RUN commands all move the actuators. The *OPC? query is used to return a response and set the MAV bit (bit 4) of the Status Byte when those commands have completed their motion(s). The STOP command will terminate all pending operations.

By enabling bit 4 of the Service Request Enable Register, the *OPC? common command can cause the generation of a GPIB <SRQ>.

The *OPC common command and the *OPC? query operate differently in how they signal an operation complete to the remote host. The *OPC? query generates a message ("1<*NL*>") when all operations are complete, which also sets the MAV bit (bit 4) in the status byte.

The *OPC command sets the OPC bit (bit 0) of the Standard Event Status Register when all pending operations are completed.

Returns: <*number*><*NL*>

<*number*> is an unsigned integer with the value one. The response is only generated when all pending operations are complete.

Examples:

```
Send:      JOG2 -10.1<NL>
Send:      *OPC?<NL>
Receive:   1<NL>
```

When the JOG2 command has finished executing, the *OPC? command will generate the response.

```
Send:      *CLS; *SRE #B00010000<NL>
Send:      JOG2 -10.1; *OPC?<NL>
Receive:   1<NL>
```

When the JOG2 command has finished executing, the *OPC? command will generate the response. The generation of the response will set the Message Available bit (bit 4) in the Status Byte. Also a GPIB <SRQ> will be generated, indicating a message is available.

Related Commands: *OPC, *WAI

3.60 *PMC Purge Macro Contents

Syntax: *PMC

Parameters: NONE

Function:

This command deletes all macro definitions. All defined macro labels and their contents are removed.

Macros are stored in dynamic memory. Removing macros releases memory for use by other commands. If a large number of macros are stored, the *PMC common command may take a short but noticeable time to remove them.

Returns: NONE

Examples:

Send:	*DMC ‘STOPXY’, “STOP1;STOP2”<*NL*>
Send:	*DMC “HOMEXY”, ‘HOME1; HOME2’<*NL*>
Send:	*LMC?<*NL*>
Receive:	“HOMEXY”,”STOPXY”<*NL*>
Send:	*PMC<*NL*>
Send:	*LMC?<*NL*>
Receive:	“”<*NL*>

Related Commands: *DMC, *EMC, *EMC?, *GMC?, *LMC?, *PMC, RMC

3.61 POLEF Set Axis 1 & 2 HCTL-1100 Pole Parameter

Syntax: POLEF [<*axis 1_polef*>][,<*axis 2_polef*>]

Parameters:

<*axis 1_polef*> is rounded to an integer <*number*> that represents the axis 1 pole filter parameter. The valid range for this value is 0 to 255. Values outside of this range should not be used.

<*axis 2_polef*> is rounded to an integer <*number*> that represents the axis 2 pole filter parameter. The valid range for this value is 0 to 255. Values outside of this range should not be used.

Function:

This command sets a servo filter parameter used by the HCTL-1100 motion controller chip on the PMC200P. This parameter effects actuator motion during a HOME, MOVE, or JOG command but has no effect during a RUN. However, the effect of the pole filter parameter on the system is nominal so any value between 20 and 150 is recommended.

Reference:

For a more technical explanation of the gain filter parameter consult a copy of the Hewlett Packard Application Note 1032.

Example:

Send:	POLEF 100<*NL*>

Sets the pole filter parameter of axis 1 to a value of 100.

Related Commands: GAINF, GAINF?, POLEF?, ZEROF, and ZEROF?

3.62 POLEF? Axis 1 & 2 HCTL-1100 Pole Parameter Query

Syntax: POLEF?

Parameters: NONE

Function:

This query returns the current setting of a servo filter parameters used by the HCTL-1100 motion controller chips on the PMC200P.

Reference:

For a more technical explanation of the gain filter parameter consult a copy of the Hewlett Packard Application Note 1032.

Returns: *<axis 1_polef>,<axis 2_polef><NL>*

<axis 1_polef>,<axis 2_polef> are of type *<number>* and represent the current settings of the pole filter parameter for axes 1 and 2.

Example:

```
Send:       POLEF?<NL>
Receive:    26,26<NL>
```

Queries the current pole filter parameter for axes 1 and 2.

Related Commands: GAINF, GAINF?, POLEF, ZEROF, and ZEROF?

3.63 POS? Axis 1 & 2 Current Position Query

Syntax: POS?

Parameters: NONE

Function:

Returns the current position of both axis 1 and axis 2 actuators, in units, relative to the origin. The POS? query is functionally equivalent to the POS1? and POS2? queries executed simultaneously.

Returns: *<axis 1_pos>,<axis 2_pos><NL>*

<axis 1_pos> is a floating point *<number>* representing the current axis 1 actuator location, in units, relative to the origin. The returned value is in standard floating point form. The factory default value is 0.00 units.

<axis 2_pos> is a floating point *<number>* representing the current axis 2 actuator location, in units, relative to the origin. The returned value is in standard floating point form. The factory default value is 0.00 units.

Examples:

```
Send:       ZERO<NL>
Send:       POS?<NL>
Receive:    000.000,000.000<NL>
Send:       MOVE -10,20<NL>
Send:       *WAI;POS?<NL>
Receive:    -010.000,020.000<NL>
```

Related Commands: POS1?, POS2?, PREPOS

3.64 POS1? Axis 1 Current Positon Query

Syntax: POS1?

Parameters: NONE

Function:

Returns the current position of the axis 1 actuator, in units, relative to the origin. While the POS? command may be used to return the current axis 1 actuator position, the POS1? command will return only the axis 1 actuator position.

Returns: *<axis 1_pos><NL>*

<axis 1_pos> is a floating point *<number>* representing the current axis 1 actuator location, in units, relative to the origin. The returned value is in standard floating point form. The factory default value is 0.00 units.

Examples:

```
Send:       ZERO1<NL>
Send:       POS1?<NL>
Receive:    000.000<NL>
Send:       MOVE1 -10<NL>
Send:       *WAI;POS1?<NL>
Receive:    -010.000<NL>
```

Related Commands: POS?, POS2?, PREPOS

3.65 POS2? Axis 2 Current Position Query

Syntax: POS2?

Parameters: NONE

Function:

Returns the current position of the axis 2 actuator, in units, relative to the origin. While the POS? command may be used to return the current axis 2 actuator position, the POS2? command will return only the axis 2 actuator position.

Returns: *<axis 2_pos><NL>*

<axis 2_pos> is a floating point *<number>* representing the current axis 2 actuator location, in units, relative to the origin. The returned value is in standard floating point form. The factory default value is 0.00 units.

Examples:

```
Send:       ZERO2<NL>
Send:       POS2?<NL>
Receive:    000.000<NL>
Send:       MOVE2 -10<NL>
Send:       *WAI;POS2?<NL>
Receive:    -010.000<NL>
```

Related Commands: POS?, POS1?, PREPOS

3.66 *PRE Parallel Poll Enable Register (GPIB only)

Syntax: *PRE <*number*>

Parameters:

<*number*> rounds to an unsigned integer in the range from 0 to 255 and is placed into the Parallel Poll Enable Register. Bits 0,1,2, and 3 are not defined by IEEE 488.2, and are don't cares.

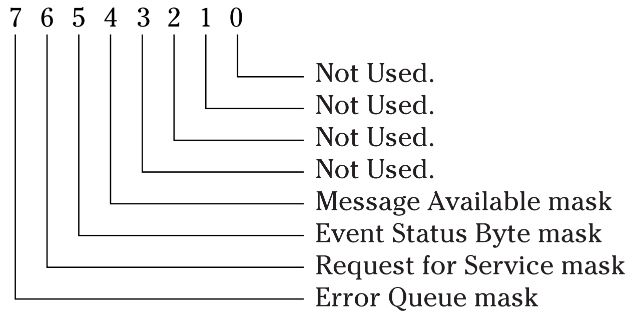
Function:

This command sets the Parallel Poll Enable Register bits. The Parallel Poll Enabled Register is used in conjunction with the Status Byte to generate the IST (Individual Status) byte.

The IEEE 488.2 standard allows the Parallel Poll Enable Register to be up to sixteen bits in size. The PMC200-P only uses and retains the low eight bits, each as a mask bit for the Status Byte.

The Parallel Poll Enable Register is preset at power-on.

The Parallel Poll Enable Register is bit mapped, with each bit signifying a different condition. The bits and their significance are detailed below:



Returns: NONE

Examples:

Send:	*PRE #B11000000<*NL*>
Send:	*PRE?<*NL*>
Receive:	192<*NL*>

If the Error Queue bit and the Request for Service bits in the Status Byte are set, then the IST is also set.

Related Commands: *IST, *PRE?

3.67 *PRE? Parallel Poll Enable Register Query (GPIB only)

Syntax: *PRE?

Parameters: NONE

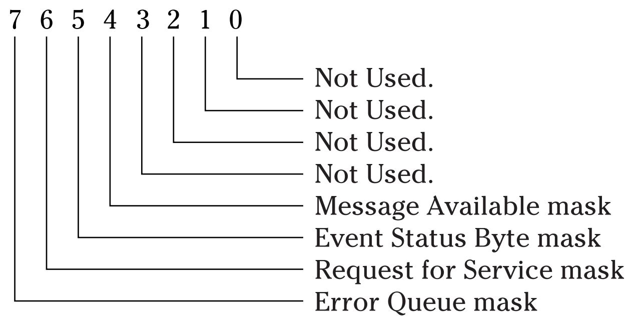
Function:

This command defines the content of the Parallel Poll Enable Register. The result is the decimal value of the mask bits.

The IEEE 488.2 standard allows the Parallel Poll Enable Register to be up to sixteen bits in size. The PMC200-P only uses and retains the low eight bits, each as a mask bit for the Status Byte.

The Parallel Poll Enable Register is preset at power-on.

The Parallel Poll Enable Register is bit mapped, with each bit signifying a different condition. The bits and their significance are detailed below:



Returns: *<number><NL>*

<number> is an unsigned integer in the range from 0 to 255 representing the bit mask value. *<number>* is always returned in decimal format.

Examples:

Send:	*PRE #B11000000*<NL>*
Send:	*PRE?*<NL>*
Receive:	192*<NL>*

If the Error Queue bit and the Request for Service bits in the Status Byte are set, then the IST is also set.

Related Commands: *IST, *PRE

3.68 PREPOS Preset Actuator Position

Syntax: PREPOS [<*axis 1_pos*>][,<*axis 2_pos*>]

Parameters:

<*axis 1_pos*> is of type <*number*>, and is evaluated as a signed floating point value. <*axis 1_pos*> defines an absolute position, in currently defined units, which is used as the actual position value for axis 1.

<*axis 2_pos*> is of type <*number*>, and is evaluated as a signed floating point value. <*axis 2_pos*> defines an absolute position, in currently defined units, which is used as the actual position value for axis 2.

Function:

This command allows a user to preset the PMC200-P system's actual position value for each actuator. This can be used to set the PMC200-P to reflect the actual position displayed on the actuator, rather than moving the actuator to the zero position and then using the ZERO command.

The PMC200-P will, after presetting the position value, perform all other commands with the new position as a basis. For example, the HOME command will use the new position to determine the zero position.

Returns: NONE

Examples:

```
Send:       POS?<NL>
Receive:    023.000,000.000<NL>
Send:       PREPOS ,100.0<NL>
Send:       POS?<NL>
Receive:    023.000,100.000<NL>
```

The PREPOS command is used, in this example, to set the actual position for axis 2 to 100.0 units. Axis 1 is not affected in this example.

```
Send:       PREPOS 25.4,180<NL>
Send:       POS?<NL>
Receive:    025.400,180.000<NL>
```

The PREPOS command is used here to set both actual position values to different values.

Related Commands: POS?, POS1?, POS2?, PREPOS?

3.69 PREPOS? Actuator Preset Position Query

Syntax: PREPOS?

Parameters: NONE

Function:

Returns the current preset position value of both axis 1 and axis 2 actuators, in units.

Returns: *<axis 1_pos>*,*<axis 2_pos><NL>*

<axis 1_pos> is a floating point *<number>* representing the current axis 1 actuator preset position value, in units, relative to the origin. The returned value is in standard floating point form. The factory default value is 0.00 units.

<axis 2_pos> is a floating point *<number>* representing the current axis 2 actuator preset position value, in units, relative to the origin. The returned value is in standard floating point form. The factory default value is 0.00 units.

Examples:

```
Send:       PREPOS 5<NL>
Send:       PREPOS?<NL>
Receive:    5.000,000.000<NL>
```

Related Commands: POS, POS1?, POS2?, PREPOS?

3.70 RMC Remove Individual Macro Contents

Syntax: RMC <name>

Parameters:

<name> is the name of a defined macro and is of type *<string>*. No case conversion is done on macro names. The name given in the parameter must conform to specifications for macro names (See *DMC). The length of the name is not restricted, but only the first 16 characters are significant. An error is generated if *<name>* is not a currently defined macro.

Function:

This command deletes the specified macro. The macro is removed from the macro list, and all memory used is returned to the free memory space maintained by the executive. The macro may be later redefined using the *DMC common command.

Returns: NONE

Examples:

```
Send:       *DMC STOPXY,"STOP1;STOP2"<NL>
Send:       *DMC HOMEXY,"HOME1; HOME2"<NL>
Send:       *LMC?<NL>
Receive:    "HOMEXY","STOPXY"<NL>
Send:       RMC HOMEXY<NL>
Send:       *LMC?<NL>
Receive:    "STOPXY"<NL>
```

Related Commands: *DMC, *EMC, *EMC?, *GMC?, *LMC?, *PMC

3.71 *RST Reset

Syntax: *RST

Parameters: NONE

Function:

This command performs a software reset, returning the PMC200-P to a known condition. The following actions are performed upon receiving a *RST command:

1. Any macros or command strings executing are aborted.
2. All foreground processes are aborted.
3. Axis 1 and Axis 2 are stopped.
4. The expansion of macros is disabled (*EMC 0).

Returns: NONE

Examples:

```
Send:      *EMC 1<NL>
Send:      *EMC?<NL>
Receive:   1<NL>
Send:      *RST<NL>
Send:      *EMC?<NL>
Receive:   0<NL>
```

The *RST command disables macro expansion in this example.

Related Commands: *CLS

3.72 RUN
Run Axis 1 & 2

Syntax: RUN [<*axis 1_vel*>][,<*axis 2_vel*>]

Parameters:

<*axis 1_vel*> converts to a floating point <*number*> that represents the axis 1 actuator velocity in units per second (units/sec). While almost any number may be entered, only values from about -300 to 300 are meaningful. The factory default value is 0. If <*axis 1_vel*> is missing, then the current axis 1 actuator velocity value is not changed. If the axis 1 actuator is in motion when the RUN command is issued, then the velocity will immediately change to the new value. The sign of the <*axis 1_vel*> parameter indicates the direction of motion when using the RUN command.

<*axis 2_vel*> converts to a floating point <*number*> that represents the axis 2 actuator velocity in units per second (units/sec). While almost any number may be entered, only values from about -300 to 300 are meaningful. The factory default value is 0. If <*axis 2_vel*> is missing, then the current axis 2 actuator velocity value is not changed. If the axis 2 actuator is in motion when the RUN command is issued, then the velocity will immediately change to the new value. The sign of the <*axis 2_vel*> parameter indicates the direction of motion when using the RUN command.

Function:

This command sets the maximum velocity for each actuator, with the velocity given in the currently defined units (units/sec). The axis 1 actuator and axis 2 actuator are then moved in the direction and speed specified by the velocity parameters or the default velocities.

The velocity specified is not mapped to the trapezoidal trajectory velocity used during a MOVE or JOG command. The trapezoidal trajectory velocities are modified when using the VEL commands, and are also set to default values when defining an actuator using the ACTTYP command.

The sign of the velocity parameters determines the direction of movement when using the RUN command. A negative sign will move the axis in the reverse direction, and a positive sign will move the axis in the forward direction.

The actual velocities for each actuator may be requested by using the AVEL? query. The RUN? query returns the currently defined maximum velocities, not the actual velocities.

This command is functionally equivalent to RUN1 and RUN2 commands being executed simultaneously.

Returns: NONE

Examples:

Send:	RUN -20.1,30<*NL*>
Send:	RUN?<*NL*>
Receive:	-020.100,030.000<*NL*>

Related Commands: RUN?, RUN1, RUN1?, RUN2, RUN2?, AVEL?, AVEL1?, AVEL2?

3.73 RUN? Axis 1 & 2 Run Velocity Query

Syntax: RUN?

Parameters: NONE

Function:

This command returns the maximum velocity for each actuator, with the velocity given in the currently defined units (units/sec). Both the axis 1 actuator and axis 2 actuator maximum velocities can be set by the RUN, RUN1, and RUN2 commands. The default value for each actuator is 0.0 units/sec. The velocity values are returned in standard signed floating point format.

The velocity specified is not mapped to the trapezoidal trajectory velocity used during a MOVE or JOG command. The trapezoidal trajectory velocities are modified when using the VEL commands, and are also set to default values when defining an actuator using the ACTTYP command.

The sign of the velocity values reflect the direction of movement when using the RUN command. A negative sign will move the axis in the reverse direction, and a positive sign will move the axis in the forward direction.

The actual velocities for each actuator may be requested by using the AVEL? query. The RUN? query returns the currently defined maximum velocities, not the actual velocities.

This command is functionally equivalent to the RUN1? and RUN2? commands being executed simultaneously.

Returns: <*axis 1_vel*><*axis 2_vel*><*NL*>

<*axis 1_vel*> is a floating point <*number*> that represents the axis 1 actuator's maximum velocity in units per second (units/sec). The returned value is in standard signed floating point format.

<*axis 2_vel*> is a floating point <*number*> that represents the axis 2 actuator's maximum velocity in units per second (units/sec). The returned value is in standard signed floating point format.

Examples:

Send:	RUN -20.1,30<*NL*>
Send:	RUN?<*NL*>
Receive:	-020.100,030.000<*NL*>

Related Commands: RUN, RUN1, RUN1?, RUN2, RUN2?, AVEL?, AVEL1?, AVEL2?

3.74 RUN1 Run Axis 1

Syntax: RUN1 [<*axis 1_vel*>]

Parameters:

<*axis 1_vel*> converts to a floating point <*number*> that represents the axis 1 actuator velocity in units per second (units/sec). While almost any number may be entered, only values from about -300 to 300 are meaningful. The factory default value is 0. If <*axis 1_vel*> is missing, then the current axis 1 actuator velocity value is not changed. If the axis 1 actuator is in motion when the RUN command is issued, then the velocity will immediately change to the new value. The sign of the <*axis 1_vel*> parameter indicates the direction of motion when using the RUN command.

Function:

This command sets the maximum velocity at which the axis 1 actuator will be driven, in units per second (units/sec). The axis 1 actuator is then moved along the axis in the direction and speed specified by the <*axis 1_vel*> parameter. If the parameter is missing, the last defined velocity for axis 1 is used.

The velocity specified is not mapped to the trapezoidal trajectory velocity used during a MOVE or JOG command. The trapezoidal trajectory velocities are modified when using the VEL commands, and are also set to default values when defining an actuator using the ACTTYP command.

The sign of the <*axis 1_vel*> parameter determines the direction of movement when using the RUN1 command. A negative sign will move the actuator in the reverse direction, and a positive sign will move the axis in the forward direction.

The actual velocities for the actuator may be requested by using the AVEL1? query. The RUN? query returns the currently defined maximum velocities, not the actual velocities.

RUN1 is used when axis 1 requires a run type of motion without activating axis 2 in this mode simultaneously.

Examples:

```
Send:      RUN1 -20.1<NL>
Send:      RUN1?<NL>
Receive:   -020.100<NL>
```

The current velocity for the axis 1 actuator is changed to -20.1 units per second and the axis 1 actuator is started moving in that direction at the specified speed.

```
Send:      RUN1 -20.1<NL>
Send:      AVEL1?<NL>
Receive:   -005.340<NL>
```

The current velocity for the axis 1 actuator is changed to -20.1 units per second and the axis 1 actuator is started moving in that direction at the specified speed. On requesting the actual velocity, it shows that the actuator is only capable of maintaining a speed of –5.34 units per second.

Related Commands: RUN, RUN?, RUN1?, RUN2, RUN2?, AVEL?, AVEL1?, AVEL2?

3.75 RUN1? Axis 1 Run Velocity Query

Syntax: RUN1?

Parameters: NONE

Function:

This command returns the maximum velocity for axis 1, with the velocity given in the currently defined units (units/sec). The axis 1 actuator maximum velocity can be set by the RUN or RUN1 commands. The default value for the actuator is 0.0 units/sec. The value is returned in standard floating point form.

The velocity specified is not mapped to the trapezoidal trajectory velocity used during a MOVE or JOG command. The trapezoidal trajectory velocities are modified when using the VEL commands, and are also set to default values when defining an actuator using the ACTTYP command.

The sign of the velocity value reflects the direction of movement when using the RUN command. A negative sign will move the axis in the reverse direction, and a positive sign will move the axis in the forward direction.

The actual velocity may be requested by using the AVEL? or AVEL1? queries. The RUN? or RUN1? queries returns the currently defined maximum velocities, not the actual velocities.

Returns: *<axis 1_vel><NL>*

<axis 1_vel> is a floating point *<number>* that represents the axis 1 actuator's maximum velocity in units per second (units/sec). The returned value is in standard signed floating point format.

Examples:

Send:	RUN -20.1,30*<NL>*
Send:	RUN1?*<NL>*
Receive:	-020.100*<NL>*

Related Commands: RUN, RUN?, RUN1, RUN2, RUN2?, AVEL?, AVEL1?, AVEL2?

3.76 RUN2 Run Axis 2

Syntax: RUN2 [<*axis 2_vel*>]

Parameters:

<*axis 2_vel*> converts to a floating point <*number*> that represents the axis 2 actuator velocity in units per second (units/sec). While almost any number may be entered, only values from about -300 to 300 are meaningful. The factory default value is 0. If <axis 2_vel> is missing, then the current axis 2 actuator velocity value is not changed. If the axis 2 actuator is in motion when the RUN command is issued, then the velocity will immediately change to the new value. The sign of the <axis 2_vel> parameter indicates the direction of motion when using the RUN command.

Function:

This command sets the maximum velocity at which the axis 2 actuator will be driven, in units per second (units/sec). The axis 2 actuator is then moved along the axis in the direction and speed specified by the <*axis 2_vel*> *parameter. If the parameter is missing, the last defined velocity for axis 2 is used.*

The velocity specified is not mapped to the trapezoidal trajectory velocity used during a MOVE or JOG command. The trapezoidal trajectory velocities are modified when using the VEL commands, and are also set to default values when defining an actuator using the ACTTYP command.

The sign of the <*axis 2_vel*> parameter determines the direction of movement when using the RUN2 command. A negative sign will move the actuator in the reverse direction, and a positive sign will move the axis in the forward direction.

The actual velocities for the actuator may be requested by using the AVEL2? query. The RUN? query returns the currently defined maximum velocities, not the actual velocities.

RUN2 is used when axis 2 requires a run type of motion without activating axis 1 in this mode simultaneously.

Returns: NONE

Examples:

```
Send:      RUN2 -20.1<NL>
Send:      RUN2?<NL>
Receive:   -020.100<NL>
```

The current velocity for the axis 2 actuator is changed to -20.1 units per second and the axis 2 actuator is started moving in that direction at the specified speed.

```
Send:      RUN2 -20.1<NL>
Send:      AVEL2?<NL>
Receive:   -005.340<NL>
```

The current velocity for the axis 2 actuator is changed to -20.1 units per second and the axis 2 actuator is started moving in that direction at the specified speed. On requesting the actual velocity, it shows that the actuator is only capable of maintaining a speed of 5.34 units per second.

Related Commands: RUN, RUN?, RUN1, RUN1?, RUN2?, AVEL?, AVEL1?, AVEL2?

3.77 RUN2? Axis 2 Run Velocity Query

Syntax: RUN2?

Parameters: NONE

Function:

This command returns the maximum velocity for axis 2, with the velocity given in the currently defined units (units/sec). The axis 2 actuator maximum velocity can be set by the RUN or RUN2 commands. The default value for the actuator is 0.0 units/sec. The value is returned in standard floating point form.

The velocity specified is not mapped to the trapezoidal trajectory velocity used during a MOVE or JOG command. The trapezoidal trajectory velocities are modified when using the VEL commands, and are also set to default values when defining an actuator using the ACTTYP command.

The sign of the velocity value reflects the direction of movement when using the RUN command. A negative sign will move the axis in the reverse direction, and a positive sign will move the axis in the forward direction.

The actual velocity may be requested by using the AVEL? or AVEL2? queries. The RUN? or RUN2? queries returns the currently defined maximum velocities, not the actual velocities.

Returns: <*axis 2_vel*><*NL*>

<*axis 2_vel*> is a floating point <*number*> that represents the axis 2 actuator's maximum velocity in units per second (units/sec). The returned value is in standard signed floating point format.

Examples:

Send:	RUN -20.1,30<*NL*>
Send:	RUN2?<*NL*>
Receive:	-030.000<*NL*>

Related Commands: RUN, RUN?, RUN1, RUN1?, RUN2, AVEL?, AVEL1?, AVEL2?

3.78 SCALE Scale Factor

Syntax: SCALE [<*axis 1_scale*>][,<*axis 2_scale*>]

Parameters:

<*axis 1_scale*> is of type <*number*>, and is interpreted as an unsigned floating point value, with a valid range of 0.001 to 9.999. <*axis 1_scale*> defined the scaling factor for axis 1.

<*axis 2_scale*> is of type <*number*>, and is interpreted as an unsigned floating point value, with a valid range of 0.001 to 9.999. <*axis 2_scale*> defined the scaling factor for axis 2.

Function:

The SCALE command defines a scaling factor for any or all axes. The scaling factor modifies, motion increment and velocity values by this multiplier. This is useful in applications where an actuator is driving a component through a pivoting armature or wedge block and the user wishes to command motion and velocities directly in convenient units. For example, in the case of a 45° wedge block (such as used in a Newport Model 416 Vertical Translation Stage or a 561-YZ Single Mode Positioner's Z-Axis) which reduces ultimate motion by one-half, a scaling factor of 2 should be chosen. A command to move 1 mm would result in a display change of 1 mm, an actual actuator increment of 2 mm, and a motion of the element driven by the wedge block of 1 mm— as desired. The default scaling factor for all axes is 1.0.

A defined scaling factor will stay in effect for a given axis until a new scaling factor is entered. The scaling factor is reset to 1.00 when device type is selected using the ACTTYP command.

Returns: NONE

Examples:

```
Send:       SCALE .1<NL>
```

In this example, all motion for axis 1 will be scaled by a factor of 1/10. Axis 2 is unaffected.

Related Commands: SCALE?

```
Send:       SCALE .1<NL>
Send:       SCALE?<NL>
Receive:    000.100,001.000<NL>
```

In this the SCALE? query is used to demonstrate that the scale factor for axis 1 was set to 1/10 while axis 2 was unaffected.

Related Commands: SCALE?

3.79 SCALE?
Scale Factor Query

3.80 SMPL Set Axis 1 & 2 HCTL-1100 Sampling Period

Syntax: SMPL [<*axis 1_smpl*>][,<*axis 2_smpl*>]

Parameters:

<*axis 1_smpl*> is rounded to an integer <*number*> that represents the axis 1 sample period in micro seconds.

<*axis 2_smpl*> is rounded to an integer <*number*> that represents the axis 2 sample period in micro seconds.

The range for these parameters is 128 to 2048, 2048 being the default value.

Function:

This command sets the sample period used by the HCTL-1100 motion controller chip on the PMC200P. This parameter can effect velocity regulation and acceleration regulation which in turn effects actuator motion during a HOME, MOVE, JOG, or RUN.

The default sample period of 2048 micro seconds allows for the lowest possible velocities and accelerations. More consistent velocity and acceleration regulation may be achieved, where minimal velocity is not so important, by decreasing the sample period. The actual effect on the minimum actual velocity achieved will depend on the configuration of your setup.

Reference:

For a more technical explanation of the gain filter parameter consult a copy of the Hewlett Packard Application Note 1032.

Example:

Send: SMPL 1500<*NL*>

Sets the sample period of HCTL-1100 chip controlling axis 1 to 1500 micro seconds.

Related Commands: SMPL?

3.81 SMPL Axis 1 & 2 HCTL-1100 Sampling Period Query

Syntax: SMPL?

Parameters: NONE

Function:

This query returns the sample period used by the HCTL-1100 motion controller chip on the PMC200P. This parameter can effect velocity regulation and acceleration regulation which in turn effects actuator motion during a HOME, MOVE, JOG, or RUN.

The default sample period of 2048 micro seconds allows for the lowest possible velocities and accelerations. More consistent velocity and acceleration regulation may be achieved, where minimal velocity is not so important, by decreasing the sample period. The actual effect on the minimum actual velocity achieved will depend on the configuration of your setup.

Reference:

For a more technical explanation of the gain filter parameter consult a copy of the Hewlett Packard Application Note 1032.

Returns: *<axis 1_smpl>*,*<axis 2_smpl>*

<axis 1_smpl>,*<axis 2_smpl>* are of type *<number>* and represent the current settings of the sample period for axes 1 and 2.

Example:

Send:	SMPL?*<NL>*
Receive:	2048,2048*<NL>*

Queries the current sample periods for axes 1 and 2.

Related Commands: SMPL

3.82 *SRE Service Request Enable (GPIB only)

Syntax: *SRE <*number*>

Parameters:

<*number*> rounds to an unsigned integer in the range from 0 to 255 and is placed into the Service Request Enable Register. Bit 6 of the <*number*> is not used by the Service Request Enable Register and it is ignored and set to zero. The power-up default is 48.

Function:

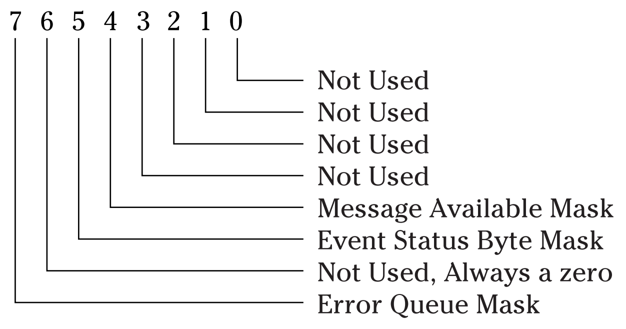
This command sets the Service Request Enable Register bits. The Service Request Enabled Register is used in conjunction with the Status Byte to generate IEEE 488.1 Service Requests.

The Status Byte Register is used to record current system conditions for the status reporting system. The register is bit mapped, each condition represented by a bit. When a bit is set, or has a value of one, then the condition is true. The bits are cleared based on the conditions described for each bit in the appendix.

The Service Request Enable Register is used to define the conditions that will generate a IEEE 488.1 <SRQ>. Bit 6 in the Service Request Enable Register is not used and is always read as a zero. If a bit is set in the Status Byte Register and its' corresponding bit is set in the Service Request Enable Register, then a <SRQ> will be generated. When a <SRQ> is generated, bit 6 in the Status Byte Register is set.

The Service Request Enable Register is configured by using the *SRE common command.

The Service Request Enable Register is bit mapped, with each bit signifying a different condition. The bits are listed below, most significant bit first:



Returns: NONE

Examples:

Send:	*SRE #B10000000<*NL*>
Send:	*SRE?<*NL*>
Receive:	128<*NL*>

The generation of an error entered into the error queue will cause a <SRQ> to be requested.

Related Commands: *SRE?

3.83 *SRE? Service Request Enable Query (GPIB Only)

Syntax: *SRE?

Parameters: NONE

Function:

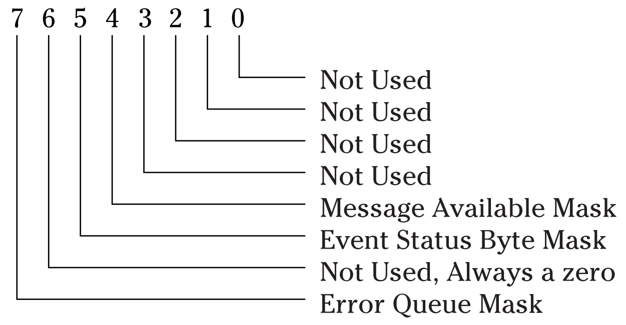
This query returns the value stored in the Service Request Enable Register.

The Status Byte Register is used to record current system conditions for the status reporting system. The register is bit mapped, each condition represented by a bit. When a bit is set, or has a value of one, then the condition is true. The bits are cleared based on the conditions described for each bit in the appendix.

The Service Request Enable Register is used to define the conditions that will generate a IEEE 488.1 <SRQ>. Bit 6 in the Service Request Enable Register is not used and is always read as a zero. If a bit is set in the Status Byte Register and its' corresponding bit is set in the Service Request Enable Register, then a <SRQ> will be generated. When a <SRQ> is generated, bit 6 in the Status Byte Register is set.

The Service Request Enable Register is configured by using the *SRE common command.

The Service Request Enable Register is bit mapped, with each bit signifying a different condition. The bits are listed below, most significant bit first:



Returns: <*number*><*NL*>

<*number*> in an unsigned integer <*number*> in the range from 0 to 255, except that bit 6 is always a zero. The return <*number*> is always in binary format. The value is defined by the *SRE common command.

Examples:

Send:	*SRE #B10000000<*NL*>
Send:	*SRE?<*NL*>
Receive:	128<*NL*>

If an error is generated and entered into the error queue, a IEEE 488.1 <SRQ> will be generated.

Related Commands: *SRE

3.84 *STB? Status Byte Query

Syntax: *STB?

Parameters: NONE

Function:

This command returns the contents of the Status Byte. The query will not modify the contents of the Status byte, except for the MAV bit (bit 4). If the query is executed from the GPIB port, the generation of the query will subsequently set the Message Available bit (bit 4) after the response has been generated.

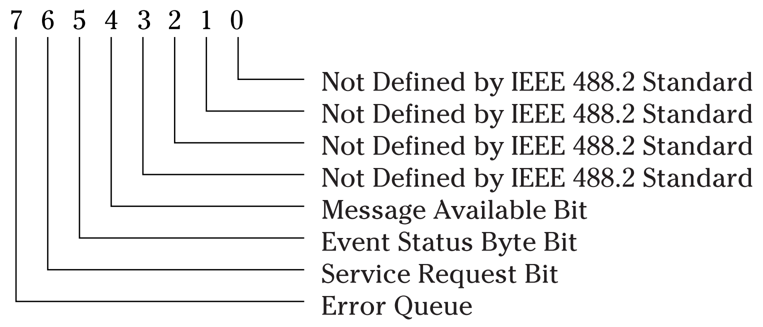
The Status Byte Register is used to record current system conditions for the status reporting system. The register is bit mapped, each condition represented by a bit. When a bit is set, or has a value of one, then the condition is true. The bits are cleared based on the conditions described for each bit.

The *CLS common command clears some of the bits in the Status Byte. The appendix describes the Status Byte in detail.

The Service Request Enable Register is used to define the conditions that will generate a IEEE 488.1 <SRQ>. If a bit is set in the Status Byte Register and its' corresponding bit is set in the Service Request Enable Register, then a <SRQ> will be generated.

The Service Request Enable Register is configured by using the *SRE common command.

The Status Byte Register is bit mapped, with each bit signifying a different condition. The bits are listed below, most significant bit first:


7 6 5 4 3 2 1 0
Not Defined by IEEE 488.2 Standard
Not Defined by IEEE 488.2 Standard
Not Defined by IEEE 488.2 Standard
Not Defined by IEEE 488.2 Standard
Message Available Bit
Event Status Byte Bit
Service Request Bit
Error Queue

Returns: *<number><NL>*

<number> is an unsigned integer in the range from 0 to 255 representing the value of the Status Byte Register. The return value is always given in decimal format.

Examples:

Send:	*STB?<*NL*>
Receive:	0<*NL*>

No bits are set in the Status Byte.

Send:	*STB?<NL>
Receive:	192<*NL*>

The Error Queue bit is set and the PMC200-P has requested a GPIB Service Request <SRQ>.

Related Commands: NONE

3.85 STOP Stop Axis 1 & 2 Motion

Syntax: STOP

Parameters: NONE

Function:

The STOP command aborts any motion for axis 1 and axis 2. The axis 1 actuator and axis 2 actuator are halted and held to the positions they were at when the STOP command took effect. The STOP command aborts all motion initiated by any command or macro.

The STOP command will also terminate all pending operations. If there were any active *OPC or *OPC? commands, these will react as if pending operations had terminated normally. If there was an active *WAI command, the STOP command will not be executed until all pending operations are complete.

Returns: NONE

Examples:

```
Send:     MOVE 350.0, 1.0;STOP<NL>
```

Will cause almost no motion since the STOP command will cancel the motion initiated by the MOVE command almost immediately.

```
Send:       MOVE 20,20;*WAI;STOP<NL>
Send:       POS?<NL>
Receive:    020.000,020.000<NL>
```

Motion is initiated on both actuators, and then a *WAI command suspends processing any other commands until motion is complete. The POS? query therefore returns the position requested, even though there was a STOP command in the input queue.

Related Commands: STOP1, STOP2

3.86 STOP1 Stop Axis 1 Motion

Syntax: STOP1

Parameters: NONE

Function:

The STOP1 command aborts any motion for the axis 1 actuator. The axis 1 actuator will be held to the position it was at when the STOP1 command took effect. The STOP1 command aborts all axis 1 actuator motion initiated by any command or macro. The axis 2 actuator is not affected.

The STOP1 command will terminate any pending operation for the axis 1 actuator. If an active *OPC command or *OPC? query is waiting on axis 1, these will respond as if pending operations have terminated normally. An active *WAI will prevent the STOP1 command from being executed until all pending operations are complete.

Returns: NONE

Examples:

```
Send:     MOVE 350.0,10<NL>
Send:     STOP1<NL>
```

Almost no motion is allowed on axis 1, since the STOP1 command will cancel the axis 1 actuator motion started by the MOVE command once the command is executed. The axis 2 actuator motion is unaffected by the STOP1 command.

Related Commands: STOP, STOP2

3.87 STOP2 Stop Axis 2 Motion

Syntax: STOP2

Parameters: NONE

Function:

The STOP2 command aborts any motion for the axis 2 actuator. The axis 2 actuator will be held to the position it was at when the STOP2 command took effect. The STOP2 command aborts all axis 2 actuator motion initiated by any command or macro. The axis 1 actuator is not affected.

The STOP2 command will terminate any pending operation for the axis 2 actuator. If an active *OPC command or *OPC? query is waiting on axis 2, these will respond as if pending operations have terminated normally. An active *WAI will prevent the STOP2 command from being executed until all pending operations are complete.

Returns: NONE

Examples:

```
Send:     MOVE 350.0,10<NL>
Send:     STOP1<NL>
```

Almost no motion is allowed on axis 2, since the STOP2 command will cancel the axis 2 actuator motion started by the MOVE command once the command is executed. The axis 1 actuator motion is unaffected by the STOP2 command.

Related Commands: STOP, STOP1

3.88 SYSDEF System Defaults

Syntax: SYSDEF

Parameters: NONE

Function:

The SYSDEF command is used to set all system variables to a known state, and also set the PMC200-P into an idle state. SYSDEF can be used whenever the user wishes to “reset” the PMC200-P without an actual hardware reset.

When SYSDEF is received the PMC200-P performs a reset of operational parameters. When finished processing the SYSDEF command, the system will be in the following state:

1. No actuator defined on any axis.
2. All remote IO parameters, such as velocity, jog displacement, etc., are set to defaults.
3. Scaling factor set to 1 to 1.
4. Display Preset cleared.
5. Backlash factor cleared.

In addition, control is returned to the PMC200-P keyboard. The GPIB and RS-232 ports will continue to accept queries, but no commands are accepted when the keyboard is in control of the PMC200-P.

Returns: NONE

Examples: NONE

Related Commands: NONE

3.89 TIME Set Current Time

Syntax: TIME <*hours*>:<*minutes*>[<*am_pm*>]

Parameters:

<*hours*> is of type <*number*>, and is interpreted as an unsigned integer, with a valid range of 0 to 23. If the <*am_pm*> modifier is used, the valid range is from 1 to 12. This is the current hour of the day. The <*hours*> parameter is mandatory for correct syntax.

<*minutes*> is of type <*number*>, and is interpreted as an unsigned integer in the range from 0 to 59. This is the current minute of the hour. The <*minutes*> parameter is mandatory for proper syntax.

<*am_pm*> is of type <*string*>, and when present in the command line, the time is evaluated as being either AM or PM. Valid parameter strings are "AM" or "PM". <*am_pm*> is not case-sensitive, and is an optional parameter.

Function:

The PMC200-P maintains a real-time clock. The clock is part of software, and is not battery backed, so time must be set after each power-up or hardware reset. The TIME command allows the current hour and minute to be set.

The current hour and minute may be retrieved by use of the TIME? query. The TIME? query always returns the time in military format.

Returns: NONE

Examples:

Send:	TIME 1:33pm<*NL*>
Send:	TIME?<*NL*>
Receive:	13:33<*NL*>

Sets the real-time clock to 1:33pm. Note that the TIME? query returns the current time in military format.

Related Commands: TIME?

3.90 TIME? Current Time Query

Syntax: TIME?

Parameters: NONE

Function:

The PMC200-P maintains a real-time clock. The clock is part of software, and is not battery backed, so time must be set after each power-up or hardware reset. The TIME command allows the current hour and minute to be set.

The current hour and minute may be retrieved by use of the TIME? query. The TIME? query always returns the time in military format.

Returns: <*hours*>:<*minutes*>

<*hours*> is of type <*number*>, and is an unsigned integer with a valid range of 0 to 23. This is the current hour of the day, given in military (24 hour) format.

<*minutes*> is of type <*number*>, and is an unsigned integer with a valid range of 0 to 59. This is the current minute of the hour.

Examples:

```
Send:      TIME  1:33pm<NL>
Send:      TIME?<NL>
Receive:   13:33<NL>
```

Sets the real-time clock to 1:33pm. Note that the TIME? query returns the current time in military format.

Related Commands: TIME

3.91 TO232 Transfer Control to RS-232C Port

Syntax: TO232

Parameters: NONE

Function:

This command transfers control to the RS-232C communications port. When a port has control, it may modify system parameters, command motions, and load and execute macros. Any port can request control at any time. Only one port can be in control at a time. However, the PMC200-P will respond to queries issued from any port whether or not it has control. Query responses will be returned to the port issuing the query.

Returns: None

Examples:

From RS-232 port:

```
Send:      TO232<NL>
Send:      ZERO1<NL>
Send:      JOG1 1.0<NL>
```

From GPIB Port:

```
Send:      POS1?<NL>
Receive:   00.0310<NL>
Send:      POS1?<NL>
Receive:   00.3250<NL>
Send:      ZERO1;POS1?<NL>
Receive:   00.6790<NL>
```

First the RS-232C port requests and receives control and initiates a jog motion. Then the GPIB port requests position and receives it (note results may vary). When repeated, the position query returns a different number, since motion is still proceeding. The GPIB port's final command to zero the actuator will cause a "-290, Not Current Controller" error, since it does not have control. This error can be seen by issuing a *ERR? query and reading the response.

Related Commands: TOGPIB

3.92 TOGPIB Transfer Control to GPIB (IEEE-488) Port

Syntax: TOGPIB

Parameters: NONE

Function:

This command transfers control to the GPIB communications port. When a port has control, it may modify system parameters, command motions, and load and execute macros. Any port can request control at any time. Only one port can be in control at a time. However, the PMC200-P will respond to queries issued from any port whether or not it has control. Query responses will be returned to the port issuing the query.

Returns: None

Examples:

From GPIB port:

```
Send:       TOGPIB<NL>
Send:       ZERO1<NL>
Send:       JOG1 1.0<NL>
```

From RS-232C Port:

```
Send:       POS1?<NL>
Receive:    00.0310<NL>
Send:       POS1?<NL>
Receive:    00.3250<NL>
Send:       ZERO1;POS1?<NL>
Receive:    00.6790<NL>
```

First the GPIB port requests and receives control and initiates a jog motion. The RS-232C port then requests position and receives it (note results may vary). When repeated, the position query returns a different number, since motion is still proceeding. The RS-232C port's final command to zero the actuator will result in a "-290, Not Current Controller" error, since it does not have control. This error can be seen by issuing a "*ERR?" query and reading the response.

Related Commands: TO232

3.93 *TST? Self Test Query

Syntax: *TST?

Parameters: NONE

Function:

As defined in the IEEE 488.2 standard, the *TST? common command causes the device to return a result of a self-test, indicating whether or not the unit completed the self-test without any errors. A zero response indicates no errors, while a non zero response indicates some error(s). The bits in the response data are left up to the designer to define.

In this version of the firmware, zero is always returned indicating no errors.

Returns: *<number><NL>*

<number> is an signed integer in the range from -32767 to 32767. A zero response indicates no error. A non zero response indicates an error(s) occurred. Zero is always returned in this version of the firmware.

Examples:

Send:	*TST?*<NL>*
Receive:	0*<NL>*

Related Commands: NONE

3.94 UNITS Define Units of Measurements

Syntax: UNITS [<*axis 1_units*>][,<*axis 2_units*>]

Parameters:

<*axis 1_units*> is of type <*string*>, and is used to set the units of measurement for the actuator attached to axis 1.

<*axis 2_units*> is of type <*string*>, and is used to set the units of measurement for the actuator attached to axis 2.

Function:

This command defines the units of measurement for the actuators, if any, that are currently attached to the PMC200-P on axis 1 and axis 2. The default units type is dependent on what motion type is selected for the axis. The defined motion types are LINEAR and ROTARY. The table below gives the supported units of measurement.

Units	Abbr.	Motion Type
Millimeters	mm	Linear (default)
Inches	inch	Linear
Degrees	deg	Rotary (default)
Milliradians	mrad	Rotary

The UNITS command accepts the abbreviations as valid input parameter strings, and is not case-sensitive with respect to parameters. For example, "MM" is the same as "mm". All characters in a parameter string is significant. Parameters are not interchangeable between motion types.

If a user switches units at any time, all position and velocity values are automatically updated to reflect the new units being used.

Returns: NONE

Examples:

```
Send:      UNITS?;POS?<NL>
Receive:   "mm","deg"<NL>
Receive:   025.400,180.000<NL>
Send:      UNITS "inch";POS1?<NL>
Receive:   001.000<NL>
```

In this example, The UNITS command forces axis 1 to uses inches as the unit of measurement, switching from millimeters. Note that the POS1? query returns a value of 1.0, which is the inch equivalent of 25.4 millimeters.

Related Commands: UNITS?

3.95 UNITS? Units of Measurement Query

Syntax: UNITS?

Parameters: NONE

Function:

This query returns the units of measurement defined for the actuators that are currently attached to the PMC200-P on axis 1 and axis 2. The default units type is dependent on what motion type is selected for the axis. The defined motion types are linear and rotary.

Returns: <*axis 1_units*>,<*axis 2_units*>

<*axis 1_units*> is of type <*string*>, and is used to set the units of measurement for the actuator attached to axis 1.

<*axis 2_units*> is of type <*string*>, and is used to set the units of measurement for the actuator attached to axis 2.

The table below gives the supported units of measurement.

Units	Abbr.	Motion Type
Millimeters	mm	Linear (default)
Inches	inch	Linear
Degrees	deg	Rotary (default)
Milliradians	mrad	Rotary

The UNITS? query uses the abbreviations as return data strings. Parameters are not interchangeable between motion types.

Examples:

```
Send:       UNITS?;POS?<NL>
Receive:    "mm","deg"<NL>
Receive:    025.400,180.000<NL>
```

In this example, The UNITS? query shows that axis 1 is using millimeters, and axis 2 is using degrees as units of measurement.

Related Commands: UNITS

3.96 VEL Set Axis 1 & 2 Maximum Velocity

Syntax: VEL [<*axis 1_vel*>][,<*axis 2_vel*>]

Parameters:

<*axis 1_vel*> converts to a floating point <*number*> that represents the axis 1 actuator velocity in units per second (units/sec). While almost any number may be entered, only values from about .001 to 100.0 are meaningful. The default value is dependent on the actuator defined. If <*axis 1_vel*> is missing, then the current axis 1 actuator velocity value is not changed. If the axis 1 actuator is in motion when the VEL command is issued, then the velocity will immediately change to the new value. A negative number will generate an error.

<*axis 2_vel*> converts to a floating point <*number*> that represents the axis 2 actuator velocity in units per second (units/sec). While almost any number may be entered, only values from about .001 to 100.0 are meaningful. The default value is dependent on the actuator defined. If <*axis 2_vel*> is missing, then the current axis 2 actuator velocity value is not changed. If the axis 2 actuator is in motion when the VEL command is issued, then the velocity will immediately change to the new value. A negative number will generate an error.

Function:

This command sets the maximum velocity for each actuator, with the velocity given in the currently defined units (units/sec). This command does not initiate a move on the actuator axes, but will modify the actuator velocities if the actuators are moving at the time this command is issued.

The velocity specified is mapped to the trapezoidal trajectory velocity used during a MOVE or JOG command.

The actual velocities for each actuator may be requested by using the AVEL? query. The VEL? query returns the currently defined maximum velocities, not the actual velocities.

This command is functionally equivalent to VEL1 and VEL2 commands being executed simultaneously.

Returns: NONE

Examples:

```
Send:      VEL 20.1,30<NL>
Send:      VEL?<NL>
Receive:   020.100,030.000<NL>
```

Related Commands: VEL?, VEL1, VEL1?, VEL2, VEL2?, AVEL?, AVEL1?, AVEL2?

3.97 VEL? Axis 1 & 2 Maximum Velocity Query

Syntax: VEL?

Parameters: NONE

Function:

This command returns the maximum velocity for each actuator, with the velocity given in the currently defined units (units/sec). Both the axis 1 actuator and axis 2 actuator maximum velocities can be set by the VEL, VEL1, and VEL2 commands. The default value for each actuator is predefined, and is dependent on the actuator type. The velocity values are returned in standard signed floating point format.

The maximum velocity value is used for both trapezoidal trajectory and velocity modes for a given actuator.

The actual velocities for each actuator may be requested by using the AVEL? query. The VEL? query returns the currently defined maximum velocities, not the actual velocities.

This command is functionally equivalent to the VEL1? and VEL2? commands being executed simultaneously.

Returns: *<axis 1_vel>,<axis 2_vel><NL>*

<axis 1_vel> is a floating point *<number>* that represents the axis 1 actuator's maximum velocity in units per second (units/sec). The returned value is in standard signed floating point format.

<axis 2_vel> is a floating point *<number>* that represents the axis 2 actuator's maximum velocity in units per second (units/sec). The returned value is in standard signed floating point format.

Examples:

Send:	VEL 20.1,30*<NL>*
Send:	VEL?*<NL>*
Receive:	020.100,030.000*<NL>*

Related Commands: VEL, VEL1, VEL1?, VEL2, VEL2?, AVEL?, AVEL1?, AVEL2?

3.98 VEL1 Set Axis 1 Maximum Velocity

Syntax: VEL1 <*axis 1_vel*>

Parameters:

<*axis 1_vel*> converts to a floating point <*number*> that represents the axis 1 actuator maximum velocity in units per second (units/sec). While almost any number may be entered, only values from about .001 to 100.0 are meaningful. The default value is dependent on the actuator defined. If the axis 1 actuator is in motion when the VEL1 command is issued, then the velocity will immediately change to the new value. A negative number will generate an error.

Function:

This command sets the maximum velocity at which the axis 1 actuator will be driven, in units per second (units/sec). This command does not initiate a move on the actuator axis, but will modify the actuator velocity if the actuator is moving at the time this command is issued.

The velocity specified is mapped to the trapezoidal trajectory velocity used during a MOVE or JOG command.

The actual velocities for the actuator may be requested by using the AVEL1? query. The VEL1? query returns the currently defined maximum velocity, not the actual velocity.

Returns: NONE

Examples:

```
Send:       VEL1 20.1<NL>
Send:       VEL1?<NL>
Receive:    020.100<NL>
```

The maximum velocity for the axis 1 actuator is changed to -20.1 units per second. No motion is initiated by the command.

Related Commands: VEL, VEL?, VEL1?, VEL2, VEL2?, AVEL?, AVEL1?, AVEL2?

3.99 VEL1? Axis 1 Maximum Velocity Query

Syntax: VEL1?

Parameters: NONE

Function:

This command returns the maximum velocity for axis 1, with the velocity given in the currently defined units (units/sec). The axis 1 actuator maximum velocity can be set by the VEL or VEL1 commands. The default value is dependent on actuator type. The value is returned in standard floating point form.

The maximum velocity value is used for both trapezoidal trajectory and velocity modes for a given actuator.

The actual velocity may be requested by using the AVEL? or AVEL1? queries. The VEL? or VEL1? queries returns the currently defined maximum velocities, not the actual velocities.

Returns: *<axis 1_vel><NL>*

<axis 1_vel> is a floating point *<number>* that represents the axis 1 actuator's maximum velocity in units per second (units/sec). The returned value is in standard signed floating point format.

Examples:

Send:	VEL 20.1,30*<NL>*
Send:	VEL1?*<NL>*
Receive:	020.100*<NL>*

Related Commands: VEL, VEL?, VEL1, VEL2, VEL2?, AVEL?, AVEL1?, AVEL2?

3.100 VEL2 Set Axis 2 Maximum Velocity

Syntax: VEL2 <*axis 2_vel*>

Parameters:

<*axis 2_vel*> converts to a floating point <*number*> that represents the axis 2 actuator maximum velocity in units per second (units/sec). While almost any number may be entered, only values from about .001 to 100.0 are meaningful. The default value is dependent on the actuator defined. If the axis 2 actuator is in motion when the VEL2 command is issued, then the velocity will immediately change to the new value. A negative number will generate an error.

Function:

This command sets the maximum velocity at which the axis 2 actuator will be driven, in units per second (units/sec). This command does not initiate a move on the actuator axis, but will modify the actuator velocity if the actuator is moving at the time this command is issued.

The velocity specified is mapped to the trapezoidal trajectory velocity used during a MOVE or JOG command.

The actual velocities for the actuator may be requested by using the AVEL2? query. The VEL2? query returns the currently defined maximum velocity, not the actual velocity.

Returns: NONE

Examples:

Send:	VEL2 20.1<*NL*>
Send:	VEL2?<*NL*>
Receive:	020.100<*NL*>

The maximum velocity for the axis 2 actuator is changed to -20.1 units per second. No motion is initiated by the command.

Related Commands: VEL, VEL?, VEL1, VEL1?, VEL2?, AVEL?, AVEL1?, AVEL2?

3.101 VEL2? Axis 2 Maximum Velocity Query

Syntax: VEL2?

Parameters: NONE

Function:

This command returns the maximum velocity for axis 2, with the velocity given in the currently defined units (units/sec). The axis 2 actuator maximum velocity can be set by the VEL or VEL2 commands. The default value is dependent on actuator type. The value is returned in standard floating point form.

The maximum velocity value is used for both trapezoidal trajectory and velocity modes for a given actuator.

The actual velocity may be requested by using the AVEL? or AVEL2? queries. The VEL? or VEL2? queries returns the currently defined maximum velocities, not the actual velocities.

Returns: *<axis 2_vel><NL>*

<axis 2_vel> is a floating point *<number>* that represents the axis 2 actuator's maximum velocity in units per second (units/sec). The returned value is in standard signed floating point format.

Examples:

Send:	VEL 20.1,30*<NL>*
Send:	VEL2?*<NL>*
Receive:	020.100*<NL>*

Related Commands: VEL, VEL?, VEL1, VEL1?, VEL2, AVEL?, AVEL1?, AVEL2?

3.102 *WAI Wait to Continue

Syntax: *WAI

Parameters: NONE

Function:

As defined in the IEEE 488.2 standard, the *WAI common command causes the device to wait until all pending operations are complete before processing any commands waiting in the input queue. The *WAI only affects input for the port from which it was received (either GPIB or RS-232), and input on the other port is processed normally.

Note that the *WAI command will essentially "lock out" a port until all pending operations are complete; not even a STOP command will be processed until all pending operations are complete. The *OPC command and *OPC? query differ from *WAI in this area, as they allow other commands to be processed while they are waiting for pending operations to complete.

Returns: NONE

Examples:

Send:	MOVE 20,0;*WAI;POS?<*NL*>
Receive:	20,0<*NL*>

In the above example, the PMC200-P does not return the requested position until all movement has been completed.

Send:	MOVE 20,20;*WAI;HOME;*WAI<*NL*>
Receive:	NONE

This command line moves the actuators to 20,20, waits for motion to be complete, and then sends the actuators to their defined zero positions, and waits for that action to complete.

Related Commands: *OPC, *OPC?

3.103 ZERO Set Axis 1 & 2 Origin

Syntax: ZERO

Parameters: NONE

Function:

The ZERO command defines the current axis 1 and axis 2 positions as the origin for each actuator. The origin is the zero position from which all movements are based.

The ZERO command has no affect on any movement in progress, however, the destinations for the MOVE commands will be computed based on the newly defined origins.

The zero buttons on the front panel provide the same feature, but are disabled while in remote mode.

The ZERO command is equivalent to executing the ZERO1 and ZERO2 commands simultaneously.

Returns: NONE

Examples:

```
Send:     MOVE 3.45,-123.20; *WAI<NL>
Send:     POS?<NL>
Receive:  003.450,-123.200<NL>
```

Move to a specified location and return the position when at the destination.

```
Send:     ZERO<NL>
Send:     POS?<NL>
Receive:  000.000,000.000<NL>
```

The ZERO command defines the current position as the new origin for the axis 1 and axis 2 actuators.

Related Commands: ZERO1, ZERO2

3.104 ZERO1 Set Axis 1 Origin

Syntax: ZERO1

Parameters: NONE

Function:

The ZERO1 command defines the current axis 1 actuator position as the new axis 1 actuator origin. The origin is the zero position from which all movements are based.

The ZERO1 command has no affect on any movement in progress, however, the destinations for the MOVE1 commands will be computed based on the newly defined origin.

The axis 1 actuator zero button on the front panel provides the same feature, but is disabled while in remote mode.

Returns: NONE

Examples:

```
Send:       MOVE 3.45,-123.20; *WAI<NL>
Send:       POS?<NL>
Receive:    003.450,-123.200<NL>
```

Move to a specified location and return the position when at the destination.

```
Send:       ZERO1<NL>
Send:       POS?<NL>
Receive:    000.000,-123.200<NL>
```

The ZERO1 command defines the current axis 1 actuator position as the new origin for the axis 1 actuator.

Related Commands: ZERO, ZERO2

3.105 ZERO2 Set Axis 2 Origin

Syntax: ZERO2

Parameters: NONE

Function:

The ZERO2 command defines the current axis 2 actuator position as the new axis 2 actuator origin. The origin is the zero position from which all movements are based.

The ZERO2 command has no affect on any movement in progress, however, the destinations for the MOVE2 commands will be computed based on the newly defined origin.

The axis 2 actuator zero button on the front panel provides the same feature, but is disabled while in remote mode.

Returns: NONE

Examples:

```
Send:       MOVE 3.45,-123.20; *WAI<NL>
Send:       POS?<NL>
Receive:    003.450,-123.200<NL>
```

Move to a specified location and return the position when at the destination.

```
Send:       ZERO2<NL>
Send:       POS?<NL>
Receive:    000.000,-123.200<NL>
```

The ZERO2 command defines the current axis 2 actuator position as the new origin for the axis 2 actuator.

3.106 ZEROF Set Axis 1 & 2 HCTL-1100 Zero Parameter

Related Commands: ZERO, ZERO2

Syntax: ZEROF [<*axis 1_zerof*>][,<*axis 2_zerof*>]

Parameters:

<*axis 1_zerof*> is rounded to an integer <*number*> that represents the axis 1 zero filter parameter. The valid range for this value is 0 to 255. Values outside of this range should not be used.

<*axis 2_zerof*> is rounded to an integer <*number*> that represents the axis 2 zero filter parameter. The valid range for this value is 0 to 255. Values outside of this range should not be used.

Function:

This command sets a servo filter parameter used by the HCTL-1100 motion controller chip on the PMC200P. This parameter effects actuator motion during a HOME, MOVE, or JOG command but has no effect during a RUN.

Problems with system stability can occur when the zero filter is much lower than 200 for linear actuators or 180 for rotary stages. A zero filter set too low may cause jitter in the actuators, cause the actuator to overshoot, and cause the actuator to take longer to settle. If the zero filter is too high then the position response of the actuator may be under-damped causing false limits to be reported due to the actuator stopping short of its intended destination.

Reference:

For a more technical explanation of the gain filter parameter consult a copy of the Hewlett Packard Application Note 1032.

Example:

Send: ZEROF 210<*NL*>
Sets the zero filter parameter of axis 1 to a value of 210.

Related Commands: GAINF, GAINF?, POLEF, POLEF?, and ZEROF?

3.107 ZEROF? Axis 1 & 2 HCTL-1100 Zero Parameter Query

Syntax: ZEROF?

Parameters: NONE

Function:

This query returns the current setting of a servo filter parameters used by the HCTL-1100 motion controller chips on the PMC200P.

Problems with system stability can occur when the zero filter is much lower than 200 for linear actuators or 180 for rotary stages. A zero filter set too low may cause jitter in the actuators, cause the actuator to overshoot, and cause the actuator to take longer to settle. If the zero filter is too high then the position response of the actuator may be under-damped causing false limits to be reported due to the actuator stopping short of its intended destination.

Reference:

For a more technical explanation of the gain filter parameter consult a copy of the Hewlett Packard Application Note 1032.

Returns: *<axis 1_zerof>,<axis 2_zerof><NL>*

<axis 1_zerof>,<axis 2_zerof> are of type *<number>* and represent the current settings of the zero filter parameter for axes 1 and 2.

Example:

Send:	ZEROF?*<NL>*
Receive:	225,225*<NL>*

Queries the current zero filter parameter for axes 1 and 2.

Related Commands: GAINF, GAINF?, POLEF, POLEF?, and ZEROF

Appendix A
Definitions

A.1 Definition of *<string>*

The IEEE 488.2 standard defines five different data types that contain string data. The PMC200-P recognizes all five types as <*string*>, thus any form may be used.

1. <*string*>, using double quotes.	“this is a string”
2. <*string*>, using single quotes.	‘this is a string’
3. <*string*>, using no quotes.	thisisastring
4. <*string*>, using block length.	#216this is a string
5. <*string*>, using 0 block length.	#0this is a string

A description of each type of <*string*> follows:

1. <*string*> defined using double quotes.

This is the form that most users are familiar with. A double quote indicates that a string follows, and the string is terminated by another double quote. A double quote may be embedded within the string by using two double quotes together:

Examples:

“this string contains a “” double quote”

All characters within the two outer double quotes are considered part of the string. It is an error if the string does not terminate with a double quote. The string cannot contain the New-Line or End Of Identify characters.

2. <*string*> defined using single quotes.

This form is identical to the double quoted string, with the obvious difference of using the single quote character to delineate the string contents. A single quote indicates that a string follows, and the string is terminated by another single quote. A single quote may be embedded within the string by using two single quotes together:

Examples:

‘this string contains a ‘’ single quote’

All characters within the two outer single quotes are considered part of the string. It is an error if the string does not terminate with a single quote. The string cannot contain the New-Line or End Of Identify characters.

3. <*string*> defined using no quotes.

This is a useful form for defining or using macro names. All strings using this format must start with an alphabetic character (A through Z, a through z). Strings of this type cannot have embedded blanks, as blanks are separator characters. Leading white space is ignored.

Some examples are shown below:

Sent: this is a string

Interpreted:

this	(1st string)
is	(2nd string)
a	(3rd string)
string	(4th string)

Sent: this,isastring

Interpreted:

this	(1st string)
,	(separator character)
isastring	(2nd string)

Sent: this'string

Interpreted:

this	(1st string)
ERROR	(non-terminated quoted string)

In the last two examples, the text after the word "this" is not considered part of the first string because a special character separates the two data items. In the first case, a comma separates strings. The last case generates an error due to non-termination of the string, not because there is no white space separating the two strings. If the entire text is to be considered one string, then one of the other four string formats would have to be used.

4. **<*string*> defined using block length.**

This is the most difficult of the five formats to use, but is the most reliable. The block format uses a block header before the actual string. The block header starts with a pound sign #. The next character must be a single character digit in the range of 1-9. This numeric character indicates the number of characters in the length field. The numeric characters of the length field immediately follow and define the length of the block. Finally, the actual text of the block follows the length field. Examples best show the relationship:

#15hello

The character 1 indicates that the length field contains one digit. The character 5 is the length field and indicates that there are five characters in the string.

#216this is a string

The character 2 indicates that the length field contains two digits. The characters 16 are the length field and indicate that there are sixteen characters in the string.

#500016this is a string

The character 5 indicates that the length field contains five digits. The characters 00016 are the length field and indicate that there are sixteen characters in the string.

Using this method, almost any character may be part of the string. The exceptions are the <EOI> signal, <NL> character, or <CR> character.

Many of the IEEE 488.2 common commands and some device specific commands return this format. This format is returned because the returned data may contain embedded quotes, commands parameters and other data that only this format may always return without confusion.

5. ***<string>* defined using no block length.**

   Like the format in 4 above, this string format starts with a pound sign #. The next character must be a zero 0. Thereafter, all characters are considered part of the string until an <EOI> signal or <NL> character is encountered. This format is useful for defining the text of a macro.

   Examples:

   ```
   *DMC “macro”,#0JOG 10,10;*WAI;POS?
   ```

   This command defines a macro called “macro”. The macro text to be substituted is everything after the #0. The actuators will step 10 of the current units, wait until the motion is completed, and return the current positions of each actuator..

A.2 Definition of *<number>*

The IEEE 488.2 standard defines four different types of numeric data. The PMC200-P recognizes all four types as *<number>*, thus any format may be used.

1. ***<number>* defined as floating point.**
2. ***<number>* defined as binary.**
3. ***<number>* defined as octal.**
4. ***<number>* defined as hexadecimal.**

Where necessary, integer numbers are converted to floating points. Floating point numbers are always treated as floating point. In all cases, a received number is terminated by any of the below characters:

, ; <NL> <EOI> <BLANK>

Any non-valid characters detected in any number received are considered an error in format, and an error condition will be generated in the system. There are no differences between the PMC200-P and IEEE-488.2 standard for number definition.

A description of each type of *<number>* follows:

1. ***<number>* defined as floating point.**

   Floating point numbers used by the PMC200-P are all single precision. Single precision maintains about six digits of accuracy and the range of the exponent is from about 10-38 to 10+38. Any of the following characters, as the first character of an ASCII sequence, indicates that a number is being defined:

   + - . 0 1 2 3 4 5 6 7 8 9

   A floating point number is defined as follows:

   1. Optional + - sign. This defines the sign of the number. If missing, positive is assumed.
   2. Optional 0 - 9 digits. These digits define the integer portion of the mantissa.
   3. Optional . decimal point. This defines the end of the integer portion of the mantissa, and indicates that the fractional portion of the mantissa follows.
   4. Optional 0 - 9 digits. These digits define the fractional portion of

the mantissa.

5. Optional exponent indicator, an ASCII ‘E’ or ‘e’, followed by a ‘+’ or ‘-’ (optional), followed by decimal digits.

Examples:

The numbers below all represent the value “1.2”

```
1.2
00001.20000
1.2e0
+001.2E+00000
+000.000000000000000000012e+20
120E-2
.12e1
```

The numbers below all represent the value “-1.2”

```
-1.2
-00001.2000
-1.2e+00
-0001.2e+0
-000.000000000000000000012E20
-120e-2
.12E1
```

2. <*number*> defined as binary.

The PMC200-P recognizes unsigned binary numbers in the range of 0 to 65535, decimal, or 0 to 1111111111111111 binary. Binary numbers are represented using only the digits 0 and 1. A binary number has the following format:

#B<*binary*>

Where

#B = mandatory binary number header
<binary> = binary digits (0’s or 1’s)

Note that the binary header can be upper or lower case.

Examples:

All numbers below represent the decimal value 129.

```
#B10000001
#b010000001
#b10000001
#B0000000010000001
```

3. <*number*> defined as octal.

The PMC200-P recognizes unsigned octal numbers in the range 0 to 65535 decimal, or 0 - 177777 octal. Octal numbers are represented using digits from 0 to 7. An octal number has the following format:

#Q<*octal*>

Where

#Q = mandatory octal number header
<octal> = octal digits (0 to 7)

Note that the octal header can be upper or lower case.

Examples:

All numbers below represent the decimal value 129.

```
#Q201
#q0201
#q201
#Q00000000201
```

4. ***<number>* defined as hexadecimal.**

The PMC200-P recognizes unsigned hexadecimal numbers in the range 0 to 65535 decimal, or 0 - FFFF hexadecimal. Hexadecimal numbers are represented using the digits 0 - 9 and the characters A - F. A hexadecimal number has the following format:

#H<*hexadecimal*>

Where

#H = mandatory hexadecimal number header
<hexadecimal> = hexadecimal digits (0 to 7 and A to F)

Note that the hexadecimal header can be upper or lower case.

Examples:

All numbers below represent the decimal value 127.

```
#H7f
#H007F
#h7f
#H000000007F
```

Appendix B
Error Messages

B.1 Error Codes and Descriptions

The IEEE 488.2 standard defines certain bits in the status registers as error condition flags. When an error occurs, one of the error bits is set in the status registers. The bit enable masks and the service request enable allow the PMC200-P to alert the remote controller that an error has occurred. The standard allows error numbers from -100 to -499.

When using the RS-232C port the Input Echo Mode controls when error are returned. When the Input Echo Mode is enabled the errors are returned immediately. When the Input Echo Mode is disabled the errors are not returned immediately and the *ERR? command must be used to retrieve the errors.

Command Errors:

Command Errors are associated with the conversion of the data received into the commands and their parameters. Incorrect syntax, incorrect parameters and improper command format will generate these errors. Any command error will cause the Command Error bit (bit 5) in the Standard Event Status Byte to be set when using GPIB.

-102,"Syntax error"

Syntax error is generated when an error in command structure or parameter type is detected. This error can be generated for any of the following conditions:

- Using a number as a command mnemonic.
- Using the wrong parameter type (string instead of number, for example).
- Using ASCII characters outside of a string constant that are not defined by the command language syntax.
- Missing parameters.
- Using parameters for commands that don't use them.

The above list in not all inclusive, but does give the user a basic idea of what to look for when this error is generated.

This error will set the Command Error bit (bit 5) in the Standard Event Status Register.

-180,"String Parameter Syntax Error"

This error is generated when the system parser detects an error in the syntax of a parameter. Similar to the data out of range error but specifically for string parameters. Sets the Command Error bit (bit 5).

-151,"Invalid string data"

This error is usually caused be an error in string termination. The IEEE 488.2 standard defines all quoted strings as having a closing quote. If the closing quote is missing, usually due to a <NL> character being embedded in the string, this error will be generated.

This error will set the Command Error bit (bit 5) in the Standard Event Status Register.

-121,"Invalid character in number"

This error indicates that a numeric parameter contains invalid data. Only one sign is allowed in the mantissa and only at the beginning of the mantissa. Only one decimal point is allowed and only in the mantissa. Only one sign is allowed in the exponent and only at the beginning of the exponent. In addition, the only characters allowed in a number are:

0-9 e E . + -

When using the binary number format, the only characters allowed are:

0 1

When using the octal number format, the only characters allowed are:

0 1 2 3 4 5 6 7

When using the hexadecimal number format, the only characters allowed are:

0-9 A-F a-f

This error will set the Command Error bit (bit 5) in the Standard Event Status Register.

-161,"Invalid Block data"

This error indicates that an invalid block header was detected. When using <string> data defined using the block length format, the block length portion of the header may only contain the digit characters, 0-9.

This error will set the Command Error bit (bit 5) in the Standard Event Status Register.

-160,"Block Data Error"

This error is generated when either a GPIB <*EOI*> signal or the <*NL*> character are embedded within a <*string*> defined using the block length format.

This error will set the Command Error bit (bit 5) in the Standard Event Status Register.

Execution Errors:

Execution Errors are associated with the interpretation of the converted commands and parameters received. Incorrect parameter values and numeric range errors are types of execution errors. Any execution error will cause the Execution Error bit (bit 4) in the Standard Event Status Byte to be set.

-222,"Data out of range"

A command required a <*number*> parameter and the <*number*> supplied was not in the allowable range. The value may have been too small, too large, or not one of the fixed values. See the command reference for the parameter ranges for each command. String parameter "range" errors usually generate a specific error for any given command.

This error will set the Execution Error bit (bit 4) in the Standard Event Status Register.

-280,"Unknown actuator type"

When defining an actuator to be used, this error is generated if the actuator specified is not supported by the PMC200-P. Valid actuator types are listed here:

850
495
496
SPECIAL
NONE

The "SPECIAL" type is always a linear actuator, similar to the 850 linear actuator, but with different gear ratios and/or drive motors.

This error will set the Execution Error bit (bit 4) in the Standard Event Status Register.

-281,"Unknown unit type"

When defining units to be used for measurement of position and velocity, this error is generated if the specified units are unknown to the PMC200-P. Valid units are listed here:

MM	(millimeters)
IN	(inches)
DEG	(degrees)
MR	(milliradians)

The units can always be expressed in either lower or upper case.

This error will set the Execution Error bit (bit 4) in the Standard Event Status Register.

-282,"Unknown actuator motor type"

When defining the motor type for a special actuator, this error is generated if a motor type unsupported by the PMC200-P is defined. Valid motor types currently supported are listed here:

1616
1624

Each of these only pertains to linear actuators. The PMC200-P currently does not support any special actuators with rotary movement.

This error will set the Execution Error bit (bit 4) in the Standard Event Status Register.

-283,"Units not compatible"

This error will set the Execution Error bit (bit 4) in the Standard Event Status Register.

-274,"Macro parameter error"

A variable parameter place-holder was defined using 0 as the parameter number. The variable parameters place-holders that may be used conform to the IEEE 488.2 standard, and are:

$1 $2 $3 $4 $5 $6 $7 $8 $9.

Place-holders greater than $9 are not supported.

This error will set the Execution Error bit (bit 4) in the Standard Event Status Register.

-279,“Active Macro Limit”

Any macro can use, as part of it’s command text, any other defined macro, including itself. This can lead to the system trying to interpret too many macros simultaneously. When this occurs, this error is generated, all current macro execution is halted, and the active macros are cleared from the system. Macro processing will still be enabled.

The macro definition below will, when the macro is executed, generate an active macro limit error, due to recursion:

```
*dmc “a”,#0move $1,$2;a;*wai
```

The macro definition below will not generate an error, but is an endless loop, requiring intervention at the PMC200-P keyboard to halt processing. This type of macro definition could be used for long term testing.

```
*dmc “a”,#0move1 $1;*wai;home1;*wai;a
```

This error will set the Execution Error bit (bit 4) in the Standard Event Status Register.

-277,“Macro redefinition not allowed”

The *DMC common command will not allow you to redefine an existing macro definition. To redefine an existing macro definition, either first remove the macro with the RMC common command, or remove all macros with the *PMC command.

This error will set the Execution Error bit (bit 4) in the Standard Event Status Register.

-278,“Macro header not found”

A macro label was used that is not defined. The *GMC? or RMC commands both require the macro name parameter to be an existing macro label.

This error will set the Execution Error bit (bit 4) in the Standard Event Status Register.

B.2 Device Errors

Device Errors are associated with system configuration and integrity. Device errors indicate a problem with the current system operation that requires immediate attention. Some device errors will set bits in the Device Event Register.

-290,“Not current controller”

This error is generated when a command is sent to the RS-232 or GPIB port when either is not the current controlling device. Note that queries will always be received and processed at any time, but only the current controlling device can process commands that will modify the PMC200-Ps configuration, or induce motion on an actuator.

This error will set bit 5 in the Device Event Register.

-291,“Command supported by GPIB only”

This error is generated only on the RS-232 port, when a command that is supported by GPIB only is sent to the RS-232 port. Commands dealing with the GPIB status reporting system (all status registers except the Device Event Register) are not supported by the RS-232 port structure.

This error does not affect the Device Event Register.

-292,"Query supported by GPIB only"

This error is generated only on the RS-232 port, when a query that is supported by GPIB only is sent to the RS-232 port. Queries dealing with the GPIB status reporting system (all status registers except the Device Event Register) are not supported by the RS-232 port structure.

This error does not affect the Device Event Register.

-320,"Insufficient memory"

This error is generated when a command that requires dynamic memory allocation cannot allocate memory. An example of such a command would be *DMC. Most often this error will occur only if there are a very large number of macros defined for the system. This error indicates a very serious condition.

This error will set bit 4 in the Device Event Register.

-330,"Actuator Limit"

This error indicates that an actuator has hit either a forward or reverse limit. This may or may not be a serious error, depending on what the actuators are being used for. The Device Event Register must be read to determine the specific limit. The error text will only be generated if the Device Event Status Enable Register mask bit is set for the specific limit.

-350 "Too Many Error"

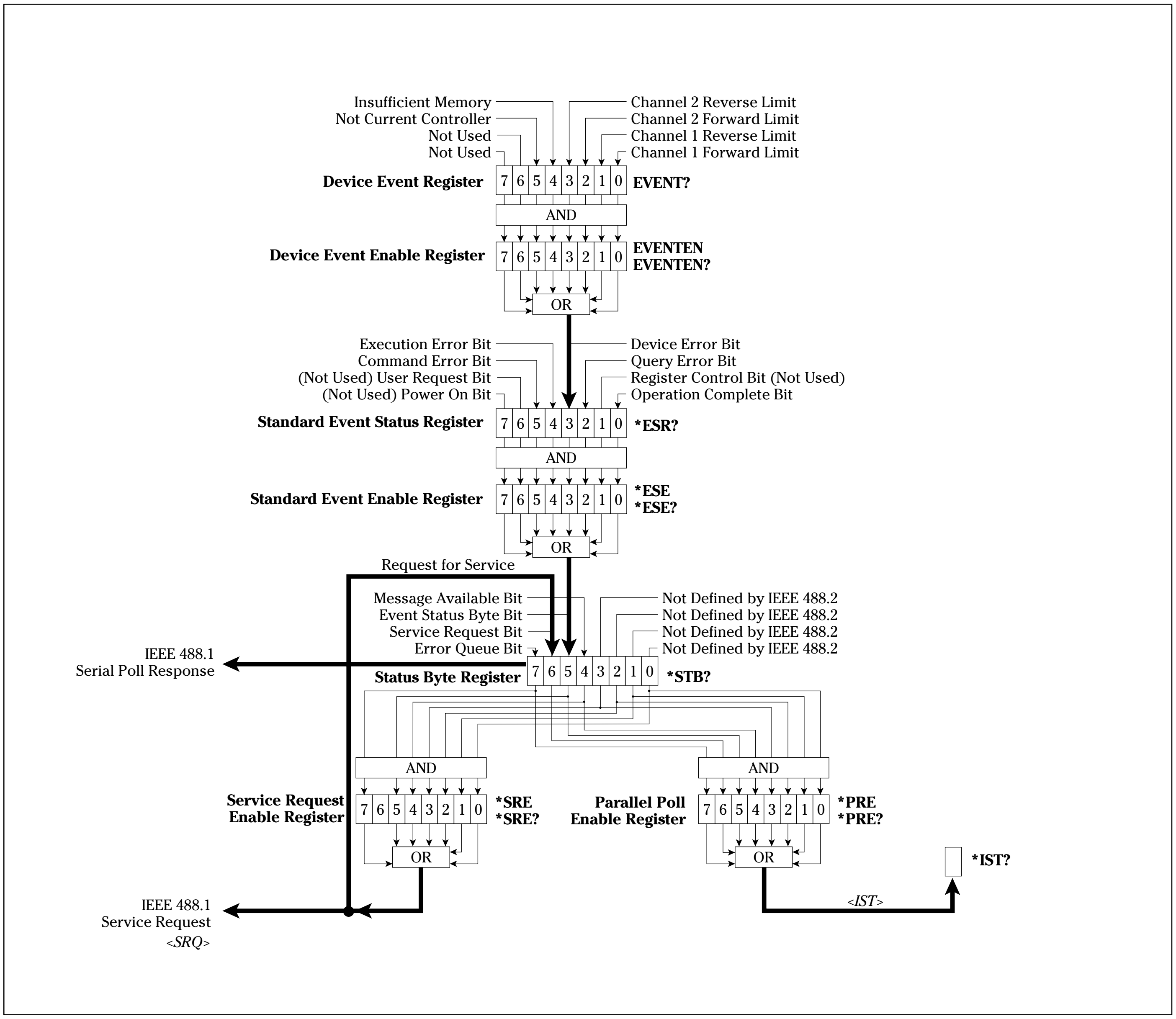
This error message is put into the queue when the queue is full. Any errors occurring after the queue is full are lost unless room is made by reading the error using the *ERR? command or by clearing the queue by using the *CLS command (GPIB only).

This error does not affect the Device Event Register.

This error will set one of the limit bits in the Device Event Register. Limit indication bits are bits 0 through 3.

Appendix C
Status Reporting System

On this next page is a graphical representation of the status reporting system for the GPIB port communications. Following, is a detailed description of each register and how the bits are set and reset. The interactions between registers is discussed and the generation of Service Requests and Parallel Polls.



Status Reporting System Flowchart

Appendix D
Register Bit Description

D.1 Device Event Register

The Device Event Register is used to record device error conditions for the status reporting system. The register is bit mapped, each condition represented by a bit. When a bit is set, or has a value of one, then the condition is true. The bit remains set until clears by either the EVENT? command or the *CLS common command.

The Device Event Enable Register is used to define the conditions that will set the Device Error bit in the Standard Event Status Register. If a bit is set in the Device Event Register and its' corresponding bit is set in the Device Event Enable Register, then Device Error bit in the Standard Event Register will be set.

The Device Event Enable Register is configured by using the EVENTEN command.

The Device Event Register is described below. The register is bit mapped, with each bit representing the described condition. The bits are listed most significant bit first.

Device Event Register.

Bit 7: Not Currently Used.
Bit 6: Not Currently Used.
Bit 5: Not Current Controller.
Bit 4: Insufficient Memory.
Bit 3: Axis 2 Forward Limit.

A one in this bit indicates that the axis 2 actuator reached a limit while travelling in the forward direction. The actuator motion was stopped upon reaching the limit. Once the Axis 2 Forward Limit bit is set, it is only cleared by the EVENT? command or the *CLS common command.

Bit 2: Axis 2 Reverse Limit.
A one in this bit indicates that the axis 2 actuator reached a limit while travelling in the reverse direction. The actuator motion was stopped upon reaching the limit. Once the Axis 2 Reverse Limit bit is set, it is only cleared by the EVENT? command or the *CLS common command.

Bit 1: Axis 1 Forward Limit.
A one in this bit indicates that the axis 1 actuator reached a limit while travelling in the forward direction. The actuator motion was stopped upon reaching the limit. Once the Axis 1 Forward Limit bit is set, it is only cleared by the EVENT? command or the *CLS common command.

Bit 0: Axis 1 Reverse Limit.
A one in this bit indicates that the axis 1 actuator reached a limit while travelling in the reverse direction. The actuator motion was stopped upon reaching the limit. Once the Axis 1 Reverse Limit bit is set, it is only cleared by the EVENT? command or the *CLS common command.

D.2 Status Event Register

The Standard Event Status Register is used to record system event conditions for the status reporting system. The register is bit mapped, each condition represented by a bit. When a bit is set, or has a value of one, then the condition is true. The bit remains set until cleared by either the *ESR? common command or the *CLS common command.

The Standard Event Enable Register is used to define the conditions that will set the Event Status Byte bit (bit 5) in the Status Byte. If a bit is set in the Standard Event Status Register and its' corresponding bit is set in the Standard Event Enable Register, then the Event Status Byte bit (bit 5) in the Standard Event Register will be set.

The Standard Event Enable Register is configured by using the *ESE common command.

The Standard Event Status Register is described below. The register is bit mapped, with each bit representing the described condition. The bits are listed most significant bit first.

Standard Event Status Register.

Bit 7: Power On.
This bit is currently not used by the PMC200-P firmware.

Bit 6: User Request.
This bit is currently not used by the PMC200-P firmware.

Bit 5: Command Error.
A one in this bit indicates that the PMC200-P has received a command on the GPIB port that generated a command error. Once the Command Error bit is set, it is only cleared by the *ESR? common command or by the *CLS common command.

Bit 4: Execution Error.
A one in this bit indicates that the PMC200-P has received a command that generated an execution error. Once the Execution Error bit is set, it is only cleared by the *ESR? common command or by the *CLS common command.

Bit 3: Device Error.
A one in this bit indicates that the PMC200-P has generated a device error. The Device Error bit is set based on the Device Event Register and the Device Event Enable Register. If any bit becomes set in the Device Event Register and its corresponding mask bit is set in the Device Event Enable Register, then the Device Error bit is set. The EVENTEN command controls the setting of the Device Event Enable Register. Once the Device Error bit is set, it is only cleared by the *ESR? common command or by the *CLS common command.

Bit 2: Query Error.
A one in this bit indicates that the PMC200-P has generated a query error. A zero in this bit indicates that no query error has been detected since the last reading or clearing of the Standard Event Status Register.

Bit 1: Request Control.
This bit is not implemented in this version of the firmware. This bit will always be a zero.

Bit 0: Operation Complete.
This bit is controlled by the *OPC common command. If the *OPC common command is in effect, then this bit will be set to one when all pending operations have been completed. A pending operation is any command what causes movement of the axis 1 actuator or axis 2 actuator. The STOP command will cause the termination of all pending operations. A zero in this bit indicates that no *OPC has been executed, or *OPC common command has been executed and pending operations have not been completed since the last reading or clearing of the Standard Event Status Register. Once the Operation Complete bit has been set, it may only be cleared by the *ESR? common command or the *CLS common command.

D.3 Status Byte Register

The Status Byte Register is used to record current system conditions for the status reporting system. The register is bit mapped, each condition represented by a bit. When a bit is set, or has a value of one, then the condition is true. The bits are cleared based on the conditions described for each bit.

The Service Request Enable Register is used to define the conditions that will generate a IEEE 488.1 <SRQ>. Bit 6 in the Service Request Enable Register is not used and is always read as a zero. If a bit is set in the Status Byte Register and its corresponding bit is set in the Service Request Enable Register, then a <SRQ> will be generated. When a <SRQ> is generated, bit 6 in the Status Byte Register is set.

The Service Request Enable Register is configured by using the *SRE common command.

The Parallel Poll Enable Register is used to define the conditions that will generate a IEEE 488.1 Parallel Poll request. If a bit is set in the Status Byte Register and its corresponding bit is set in the Parallel Poll Enable Register then <IST> is set true. If <IST> is equal to <S BIT> and the Parallel Poll Register contains a non zero value, then a Parallel Poll request will be generated.

The Parallel Poll Enable Register is configured by using the *PRE common command.

The Service Request Enable Register is configured by using the *SRE common command.

The Status Byte Register is described below. The register is bit mapped, with each bit representing the described condition. The bits are listed most significant bit first.

Status Byte.

Bit 7: Error Queue.
A one in this bit indicates that the error queue is not empty. The Error Queue stores error codes as they are detected. Query errors are always placed into the Error Queue. Once the Error Queue is emptied, then the Error Queue bit is cleared. The *CLS common command will remove all errors from the error queue and clear the Error Queue bit.

Bit 6: Service Request.
This bit is set when the status reporting system has determined that a new reason for service has occurred and a service request will be generated. The *SRE common command defines the mask bits used to generate the service request. If any bit (except the Service Request bit) becomes set, and its corresponding bit in the Service Request Enable

Register is set, then a service request will be generated and the Service Request bit will be set. This bit will become cleared when the reason for service no longer exists, or by the *CLS common command.

This bit may be masked for use by the *PRE common command for generating parallel polls.

Bit 5: Event Status Byte.
The Event Status Byte bit is set based on the condition of the Standard Event Status byte masked with the Standard Event Enable Register. When a bit in the Standard Event Status Register becomes set and its corresponding bit in the Standard Event Enable Register is set, then the Event Status Byte bit will be set. Once this bit is set, the *CLS common command may clear it.

Bit 4: Message Available.
The Message Available MAV bit becomes set when any message is ready to be transmitted over the GPIB bus. Once the message is sent, the MAV bit is cleared. Another queued message ready to be sent would then set the MAV bit. The *CLS common command has no effect on this bit.

Bit 3: Not Defined by IEEE 488.2 Standard.
Bit 2: Not Defined by IEEE 488.2 Standard.
Bit 1: Not Defined by IEEE 488.2 Standard.
Bit 0: Not Defined by IEEE 488.2 Standard.

Appendix E
Sample Programs

E.1 Example Program: PMC200-P RS232C Communication

```
10 '************************** Program Header **************************
20 'NEWPORT CORPORATION
30 'PMC200-P to RS-232 Communication Program - an example program
40 '
50 'This program was written in MCROSOFT GWBASIC and designed to show
you
60 'how to write a simple program that will write commands and read query
70 'responses to and from the PMC200-P controller and IBM PC/AT or compat-
ible.
80 'Before starting, connect a standard 9-pin cable between the PMC200-P
90 'RS-232 port and the RS-232 COM1 port of an IBM PC/AT and follow the
100 'instructions in the users manual for setting up the RS-232 port on the
110 'PMC200-P.  NOTE: Be sure that the INPUT ECHO MODE: is disabled.
120 '
130 'Written By:   Darwin D. Smith
140 '        Date:   November 1, 1990
150 '************************* End of Header ****************************
1000 'Beginning of program
1010      'Open COM port with the following specifications:
1020      'COM port 1, 9600 baudrate, no parity, 8 data bits & 1 stop bit
1030      OPEN "COM1:9600,N,8,1" FOR RANDOM AS #1
1040  GOSUB 2000                        'Draw header on the screen
1050  GOSUB 3000                        'Process user input
1060  CLOSE #1                          'Close the COM file
1070 END 'End of program
1080 '
2000 'Main.Screen: Draw the main screen.
2010  CLS
2020  'Draw header
2030  LOCATE 1, 24: PRINT "N E W P O R T   C O R P O R A T I O N"
2040  LOCATE 2, 22: PRINT "PMC200-P to RS-232 Communication Program"
2050  LOCATE 3, 33: PRINT "Enter Q to quit"
2060  LOCATE 4, 20: PRINT "Just hit <ENTER> to force read of RS-232 port"
2070  PRINT
2080 RETURN
2090'
3000 'Enter.User.Commands: Get and interpret the user's commands.
3010  WHILE (1)   'Get and process user input until Q is input.
3020      RS-232OUT$ = ""                 'Clear RS-232out$ string
3030      LINE INPUT RS-232OUT$           'Get the user input
3040      IF RS-232OUT$ = "Q" OR RS-232OUT$ = "q" THEN RETURN
3050      IF RS-232OUT$ = "" THEN GOSUB 4000 ELSE PRINT #1, RS-232OUT$
3060  WEND
3070 RETURN
3080 '
4000 'Read.RS-232.String: Read and print all characters from the RS-232 port.
```

```
4010      BUFFER$ = “”                          ‘Initiate buffer$
4020      ‘While there is a character available read it and print it
4030   WHILE (LOC(1) > 0)
4040      BUFFER$ = INPUT$(1, #1)         ‘Input one character from COM port
4050      PRINT BUFFER$;                        ‘Print the character
4060      WEND
4070 RETURN
4080 ‘
10000 END ‘End of Program Listing
```

E.2 Example Program: PMC200-P IEEE-488 Communication

```
'*************************** Program Header **************************
'NEWPORT CORPORATION
'PMC200-P to IEEE-488 Communication Program - an example program
'
'The following program is designed to be an example of how to write a simple
'program that will write commands and read query responses to and from
'the PMC200-P controller via the IEEE-488 port.  This program was specifically
'written for the equipment below.
'
'This program was written on an IBM AT compatible using QuickBASIC 4.5
'  and software routines supplied with the GPIB board.
'The GPIB board used was: National Instruments GPIB-PC2A board
'                         Newport Corporation pn LA-PC-488-2A-5
'The interface cable was:  Newport Corporation pn LA-CABLE-2M488
'
'NOTE: Before running this program on the above GPIB board follow the
'        manufacturers instructions for installation of hardware and software.
'        Be sure to read the section on setting up the QuickBASIC environment.
'        Rename one of the devices "PMC200-P" when running the IBCONF
program
'        included with the board.  Remember the device number you renamed
'        and choose that number as the IEEE address when in the GPIB set-up:
'        menu on the PMC200-P.
'
'Written By: Darwin D. Smith
'      Date:  November 1, 1990
'**************************** End of Header ***************************
'Beginning of program
'$INCLUDE: 'D:\b45\QBDECL4.BAS'      'Use your own path here.
'Beginning of program
   IEEEout$ = "PMC200-P"                'IEEE-488 PMC200-P address must be
                                        'named PMC200-P in the GPIB.COM file.
   CALL IBFIND(IEEEout$, device.number%) 'Set-up the device.number% variable
                                        'used in GPIB-PC routine calls.
                                        '(National Instruments command)
   IF device.number% < 0 THEN
      PRINT "Unable to find "; IEEEout$; " device."
   STOP
  END IF
  GOSUB Main.Screen                     'Draw header on the screen
  GOSUB Enter.User.Commands             'Process user input
END  'End of program

'Main.Screen: Draw the main screen.
Main.Screen:
   CLS
   LOCATE 1, 22: PRINT "N E W P O R T   C O R P O R A T I O N"
   LOCATE 2, 19: PRINT "PMC200-P to IEEE-488 Communication Program"
   LOCATE 3, 33: PRINT "Enter Q to quit"
   LOCATE 4, 16: PRINT "Just hit <ENTER> to force a read of the IEEE port"
   PRINT : PRINT
RETURN

'Enter.User.Commands: Get and interpret the user's commands.
Enter.User.Commands:
   DO      'Get and process user input until E, EXIT, Q, or QUIT input.
      IEEEout$ = ""                     'Clear IEEEout$ string
```

```
        LINE INPUT IEEEout$                 'Get the user input
        IEEEout$ = UCASE$(IEEEout$)         'Convert input to upper case
        SELECT CASE IEEEout$
            CASE "Q"                        'Exit on Q
                RETURN
        CASE "IBCLR"                        'Allow user to issue device clear
            CALL IBCLR(device.number%)      'Send a device clear
                                            '(National Instruments command)
        CASE ""
            GOSUB Read.IEEE.String          'Read input data on IEEE port
        CASE ELSE
            'Write IEEEout$ to IEEE-488 port using National Instruments command
            CALL IBWRT(device.number%, IEEEout$ + CHR$(10))
        END SELECT
    LOOP WHILE 1 = 1
RETURN

'Read.IEEE.String: Read a string from the PMC200-P IEEE port.  The string
'   read from the IEEE-488 port is printed on the screen.
Read.IEEE.String:
    timeout.error% = 0                      'Initiate to no time-out error
    ON TIMER(2) GOSUB timeout               'Set timer for a 2 second time-out
    IEEEin$ = SPACE$(255)                   'Initiate IEEEin$ to 255 spaces
    poll% = 0

    'wait for MAV bit to be set by doing serial polls of PMC200-P
    TIMER ON 'Turn timer on
    WHILE ((timeout.error% = 0) AND ((poll% AND &H10) = 0))
        CALL IBRSP(device.number%, poll%)
    WEND
    TIMER OFF                               'Turn timer off

    'If did not timeout keep reading IEEE port while the MAV bit is set
    IF (timeout.error% = 0) THEN
    WHILE ((poll% AND &H10) = &H10)
        CALL IBRD(device.number%, IEEEin$)  'Read IEEE-488 port
        IEEEin$ = RTRIM$(IEEEin$)           'Trim trailing spaces
        PRINT IEEEin$;                      'print the input string
        IEEEin$ = SPACE$(255)               'Re-initiate IEEEin$ to 255 spaces
        CALL IBRSP(device.number%, poll%)   'Do a serial poll
    WEND
    END IF
RETURN

'timeout: Set timeout.error% flag.  Called if t seconds, as defined by the
'       ON TIMER(t) function call, has elapsed between TIMER ON and TIMER
OFF.
timeout:
    timeout.error% = 1
    PRINT "ERROR-IEEE-488 port timed out during read."
RETURN

END  'End of Program Listing
```

Newport Corporation
Worldwide Headquarters

1791 Deere Avenue
Irvine, CA 92606

(In U.S.): 800-222-6440
Tel: 949-863-3144
Fax: 949-253-1680

Internet: sales@newport.com

Newport

Visit Newport Online at: www.newport.com