ACUTRONIC

ACUTROL3000 COMMAND LANGUAGE (ACL)

PROGRAMMING MANUAL

TM-8004-C

ACUTROL3000 COMMAND LANGUAGE (ACL)

PROGRAMMING MANUAL

TM-8004-C

Prepared For:	Acutronic
Date Printed:	12/12/2005
Date Prepared:	10/13/2003
Acutronic R&D Sales Order:	A-8901

ACUTRONIC

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 *ACUTROL3000* COMMAND LANGUAGE INTRODUCTION

The Acutrol Command Language (ACL) is new to the Acutrol®3000 motion controller and was developed as a super-set of the GPIB command protocol used in the Acutrol® Act® 2000 control system. The language includes a comprehensive set of commands to configure, calibrate, and operate an Acutrol3000 motion controller. The language attempts to be as backward compatible as possible to the Acutrol Act2000; however, this objective is not 100% achieved because of hardware, operational, and feature differences.

Legacy programs/commands that use touch sequences or touch sequence macros, will not function as expected. Such sections of software code will require re-writing.

The Acutrol Command Language is used for all non-realtime interfaces:

1. Instrumentation Interface using the IEEE-488.2 (GPIB) interface standard.
2. Operator Interface (integral or external) using Ethernet - TCP/IP protocols.
3. Reflective (Real-time) interfaces which support ACL command and response blocks (SCRAMNet+ and VMIC)
4. Future interface options using USB, Fire-wire, or other serial standards would use the ACL protocol.

2 PROGRAMMING COMMANDS

The general form of ACL commands consists of a traversal path and a list of arguments:

:[MAIN:SubGroup1..::Subgroup(n)][sp][Argument1,Argument2..,Argument(m)]

- Some commands do not require an argument list.
- Some commands do not have sub-group(s).
- Axis specific commands define the axis by using a numeric value in Argument1:

:[MAIN:SubGroup(s)][sp][Axis#,Argument2..,Argument(m)]

For most ACL interface options, all commands issued to Acutrol will receive a response regardless of which direction data is intended to flow. The response includes an error code and a message length; this is prefixed to data that had been requested by a read or query command. The IEEE488 interface protocol only receives a response if the command requests data. Inherent in the IEEE488 protocol are hardware lines that control data flow and error messaging (SRQ) not available in basic serial interfaces.

2.1 COMMAND MNEMONICS

The IEEE-488.2 standard suggests a short form for abbreviating command mnemonics to 3 or 4 characters in length. In the descriptions and examples that follow, the short form mnemonics are not specifically identified. **The Acutrol command messages are <u>not</u> case sensitive.** Commands may be in lower, upper, or a mix of both cases to make commands more readable.

Note also that the subsystem names in the Acutrol3000 ACL do not necessarily all have unique first letters in the command mnemonics, as do the Acutrol Act2000 IEEE-488 commands. This means that care must be taken when using "short-command" mnemonics that consists of only one letter. These should only be used whenever speed is more important than clarity.

The following examples all produce the same result:

EXAMPLE: :Demand:Position 3,10.5<END>

:Dem:Pos 3,10.50000<END>

:D:P 3,10.5<END>

The following examples all produce an error:

EXAMPLE: :Demand:Posn 3,10.5<END> (spelling)

:Dem;Pos 3,10.50000<END> (miss use of semicolon)

:D:P,3,10.5<END> (improper use of comma between command and data arguments)

2.2 BASIC COMMAND TYPES

The commands for this system are of two types as defined below.

2.2.1 MAIN LEVEL COMMANDS

The MAIN level commands are at the root level or base of the parsing tree. MAIN commands represent the major branches of the command tree, and are always correctly parsed if they occur at the beginning of a program message; they must be preceded by a colon ":".

EXAMPLE: :Read... (Read data)

...;:Mode... (Command servo mode in a multi-command string)

2.2.2 SUBSYSTEM COMMANDS

Subsystem commands are functionally and/or logically grouped under the MAIN level commands, allowing repeated subsystem commands under the same node without reference to the root of the tree. Sub system commands are linked using the semi-colon ";" operatoras shown in the following example.

EXAMPLE: :Mode:Position 1;Rate 2<END>

This example causes Axis1 to change to Position mode, and Axis2 to change to Rate mode. Note, since these sub commands are part of the same ACL command string, the mode changes will occur on the same ACP frame.

2.3 TREE TRAVERSAL RULES

- A command header is formed by the concatenation of legal transitions through the command tree structure.
- A compound header is made of two or more mnemonics separated by colons and containing no white space.

EXAMPLE: :Mode:Position...

The following rules are applied to parsing the tree command structure:

- The parser begins at the root of the command tree if the command string begins with a colon, or if a program message terminator has been previously encountered (<END>).

EXAMPLE: :Demand:Position 1,123.45678<END>

- When a subsystem command is executed under the root, the parser stays at that subsystem level and will execute other commands that are in the same subsystem.
- A command that is not in the current subsystem will result in a command parsing error.

In a complex command message:

- The parser location can be found by finding the last colon before the last mnemonic in the compound header.
- Any command below this point can be sent as a part of the current program string without using the preceding mnemonics
- Subsystem commands are separated with a semicolon.

EXAMPLE: :Demand:Pos 2,0.01;Rate 3,100<END>

or :DEM:POS 2,0.01<END>
:DEM:RATE 3,100<END>

EXAMPLE: :Dem:Pos 2,0.01;:Mode:Pos 2;Rate 1<END>

or :Dem:Pos 2,0.01<END>
:Mode:Pos 2<END>
:Mode:Rate1<END>

- Note that in all of these examples, the first colon is optional.

2.4 MANUAL CONVENTIONS

The following abbreviations and symbols are use throughout this manual.

[…]	The contents of the square brackets are optional.
{…|…}	Select exactly one item specified between the braces.
➔	Acutrol responds with the following data.
<LF>	ASCII line feed, hex 0A.
<CR>	ASCII carriage return, hex 0D.
<END>	EOI asserted with the last message byte. Optionally, an ASCII carriage return and/or line feed may be appended to the given string.
<axis>	String of decimal digits specifying the axis. See section 2.6.1 Generic Numeric Formats.
<n bytes of …>	This specifies a number (n) of binary bytes of the indicated data. This data is sent least significant byte first.
<…>	This specifies that data representing the indicated quantity that will be sent or received from Acutrol.
Command?	A message that ends with a question mark is a query that returns data.
Command[?]	A message than specifies a question mark in square brackets indicates that both the command and query are available.
`Courier Typeface`	This specifies the data to be sent or received from Acutrol. Notes or other comments will appear in the normal Times Roman typeface.

These additional symbols are also used throughout this manual. These symbols are to be sent and received in the device dependent messages.

:	Command separator.
;	Program message separator.
"…"	Delimits string from remainder of message.

Because of publishing restrictions, many examples extend beyond one printed line. The wrap to the next line does not imply that a line feed will be sent or received. A line feed will be

transmitted or received where indicated by a <LF> , <END>, or there is clearly no additional text in the command.

In section 4 Acutrol3000 Commands, many of the commands are axis specific and in general the axis is specified by <axis>, the first argument in the parameter list.

The <axis> parameter is an [NR1] numeric character corresponding to the axis to be addressed, but must correspond to an axis that exists in the system. A numeric axis reference is used for both commands and queries.

For commands only, "ALL" may be used for the <axis> argument causing the command to be applied to all axes that are present.

An error will result if an axis us undefined or if "ALL" is used for a query.

2.5 DEVICE DEPENDENT MESSAGES

Acutrol can receive up to 32K (32,768) characters in a single message. More than this will overflow the read buffer and the message will be lost. Acutrol can transmit up to 32K (32,768) characters of response data in a single message.

2.6 DEVICE DEPENDENT DATA FORMATS

Data is transmitted over a non-realtime interface for the purpose of controlling and/or monitoring a motion system. Data input to Acutrol is referred to as "program data". Data that is returned is referred to as "response data".

Program data can be sent to the controller in several data formats, where the data type is indicated by the Command of the message. All data is in ASCII decimal format (<command arg>, <NR1>, <NR2>, <NR3>, or <NRf>) unless the "#" prefix is used, in which case it is interpreted as binary data (<Definite Length Arbitrary Block Response Data> or an <Indefinite Length Arbitrary Block Response> for the IEEE-488 interface).

The format used for Acutrol numeric response data is specified by the :UTILity:FORMat and :UTILity:COMPatibility commands. The following commands are affected by the :UTILity:COMPatibility command. (These commands only supported the :UTILity:FORMat BINary in the Act 1000/2000.)

- :READ:SVECTor
- :READ:VECTor
- :STATus:ECP
- :STATus:MIS

- :STATus:SUPervisor
- :UTILity:SRQ?

All other commands are controlled by the setting of the :UTILity:FORMat command.

When BINary format is specified, Acutrol will respond with <Definite Length Arbitrary Block Response Data>. When FLOat is specified, Acutrol will respond with <NR1>, <NR2>, or <NR3> format data depending on whether an integer or floating point value is required respectively.

2.6.1 GENERIC NUMERIC FORMATS

2.6.1.1 NUMERIC COMMAND ARGUMENT

In many Acutrol {ACL} device dependent messages, the controller must specify the target axis or other command argument (such as selecting one of several status words). This data must be supplied in a subset of the <NR1> integer numeric response data and excludes negative numbers.

This format is a sequence of decimal digits. The Command for the <command arg> specification is shown in the following Command diagram.

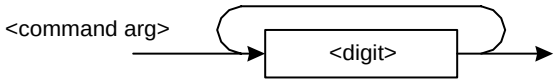


Figure 1: <command arg> Listen Command

EXAMPLE: 0 (hex 30)
1 (hex 31)
12 (hex 31, hex 32)

2.6.1.2 DECIMAL NUMERIC PROGRAM DATA

This format is referred to as <NRf> for flexible Numeric Representation. This is a listening format that accepts a wide range of decimal numeric data:

- Integer
- Floating point with and without an exponent
- Positive and negative data

Acutrol will accept this listening format for any command that requires a floating-point value. In addition, several common commands require that Acutrol accept this format.

The Command for <NRf> format data is shown in the following Command diagrams. Note that suffixes are not supported.

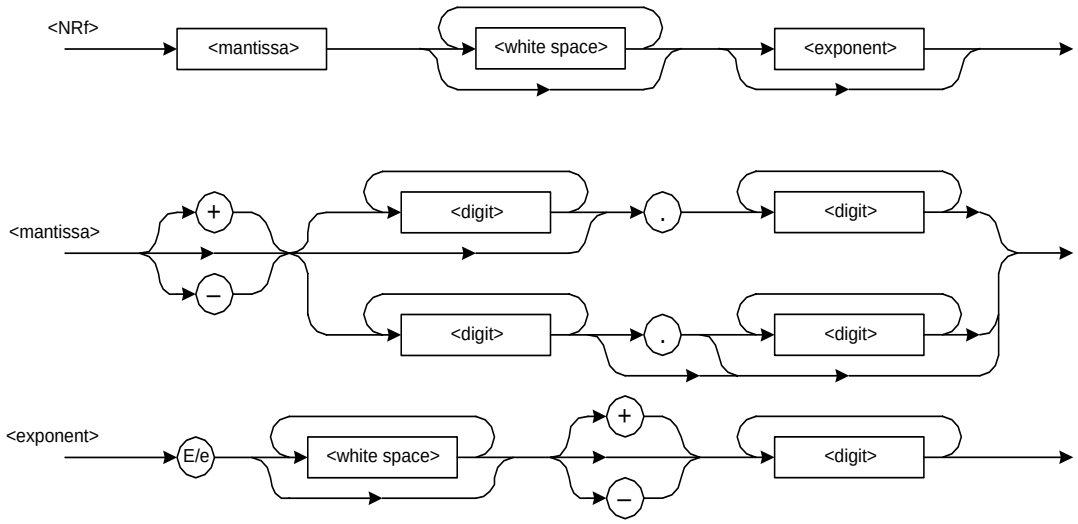


Figure 2: <NRf> Flexible Listen Command

EXAMPLE:
```
.123
  1.23
  +12.3
  .123 E  –  45
  +12.3e+45
  0.123  E  +  4
  +12.3e–4
  –.123
```

2.6.1.3 NR1 NUMERIC RESPONSE DATA

Acutrol returns integer data to the controller in <NR1> format if the variable has been configured for NR1 output using the Configure:Variable command. In addition, Acutrol also accepts this format wherever an integer value is required.

This format specifies a sequence of decimal digits preceded by an optional sign. This is shown in the following Command diagram. <NR1> format is a subset of the <NRf> format.

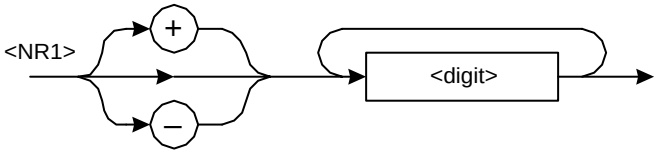


Figure 3: <NR1> Integer Response Command

EXAMPLE:
```
1
+1          Acutrol will never respond with a leading + sign.
–12
```

2.6.1.4 NR2 NUMERIC RESPONSE DATA

Acutrol returns floating point data to the controller in <NR2> format if the variable has been configured for NR2 output using the Configure:Variable command. In addition, Acutrol also accepts this format wherever a floating-point value is required.

This format specifies an optional sign followed by a sequence of decimal digits, a decimal point, and then optional digits. This is shown in the following Command diagram. <NR2> format is a subset of the <NRf> format.

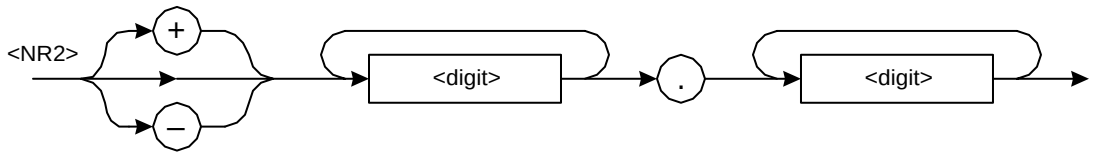


Figure 4: <NR2> Fixed Point Response Command

EXAMPLE:
```
12.3
+1.234
-0.1234
```
{Acutrol will never respond with a leading +sign}

2.6.1.5 NR3 NUMERIC RESPONSE DATA

Acutrol returns floating point data to the controller in <NR3> format if the variable has been configured for NR3 output using the Configure:Variable command. In addition, Acutrol also accepts this format wherever a floating-point value is required.

The format for <NR3> data is shown in the following Command diagram. <NR3> format is a subset of the <NRf> format.

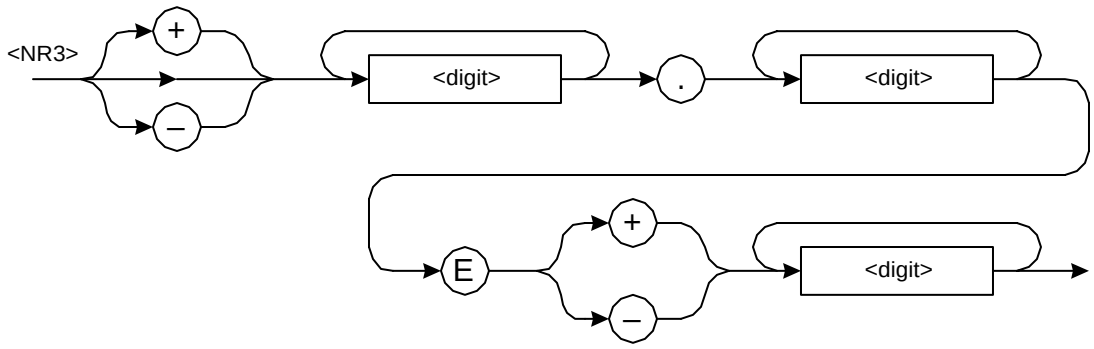


Figure 5: <NR3> Scientific Notation Response Command

EXAMPLE:
```
1.23E+45
+12.345E-67
-0.012E+3
```
{Acutrol will never respond with a leading +sign}

2.6.1.6 DEFINITE LENGTH ARBITRARY BLOCK RESPONSE DATA

Acutrol will respond with <Definite Length Arbitrary Block Response Data> whenever it must return binary data to the controller. This occurs when FORmat BINary or COMPatibility BINary is selected. In addition, Acutrol will accept data in this format in place of all numeric data ***except*** the <command arg> and <axis> specifiers. The exact meaning and scaling of this data depends on the Acutrol device dependent message.

The <non-zero digit> is a decimal digit other than 0 that specifies the number of <digits> that follow. The digits are interpreted as an integer that specifies how many <8-bit data bytes> are to follow.

The format of the <Definite Length Arbitrary Block Response Data> is shown in the following Command diagram.

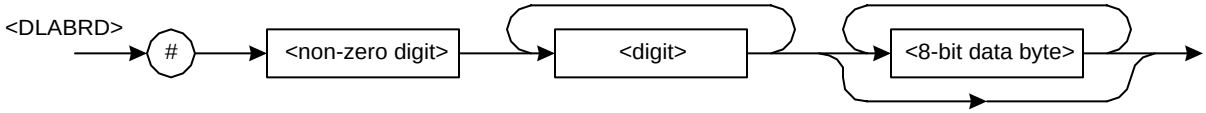


Figure 6: <Definite Length Arbitrary Block Response Data>

The following examples are all valid and equivalent.

EXAMPLE:
```
#14<byte 0><byte 1><byte 2><byte 3>
#204<byte 0><byte 1><byte 2><byte 3>
#3004<byte 0><byte 1><byte 2><byte 3>
```

This can now represent 16/32/64 bit integer data as well as single/double precision IEEE-754 floating point.

2.6.1.7 INDEFINITE LENGTH ARBITRARY BLOCK RESPONSE

The <Indefinite Length Arbitrary Block Response> may be sent to Acutrol by the controller in place of any numeric data ***except*** the <command arg> and <axis> specifiers. This format may be used in place of the <Definite Length Arbitrary Block Response Data> when it occurs at the end of a message. Acutrol will never generate an <Indefinite Length Arbitrary Block Response>.

Note: In the case of the IEEE488 protocol, the execution of an <END> is done with a hardware handshake. This is not possible with a TCP/IP protocol; consequently, Indefinite Length Arbitrary Block Response is not possible with TCP/IP and is only supported for the IEEE-488 interface.

The format of the <Indefinite Length Arbitrary Block Response> is shown in the following Command diagram.

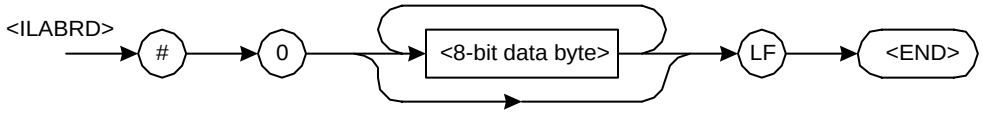


Figure 7: <Indefinite Length Arbitrary Block Response>

The following examples are equivalent.

EXAMPLE: `#0<byte 0><byte 1><byte 2><byte 3><END>`

`#14<byte 0><byte 1><byte 2><byte 3><END>`

2.6.1.8 ARBITRARY ASCII RESPONSE

Several Acutrol device dependent commands require the use of <Arbitrary ASCII Response> data (AARD). The Acutrol response to a query for these commands also returns data formatted as <Arbitrary ASCII Response> data.

The format of the <Arbitrary ASCII Response> data is shown in the following Command diagram.

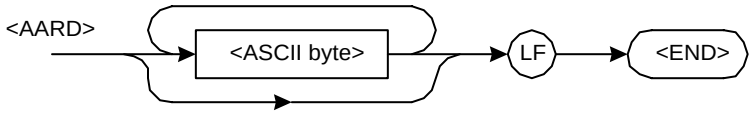


Figure 8: Arbitrary ASCII Response Data

2.6.1.9 HEXADECIMAL NUMERIC RESPONSE

Acutrol returns Hexadecimal data to the controller if the variable has been configured for H8, H16, H32, H64 output using the Configure:Variable command. The number of characters returned is 2, 4, 8, or 16 for the respective formats.

Figure 10: Hexidecimal Response Data

Examples of data output for various word widths.

read:var 1238 ➔ #H8B (H8)

read:var 1238 ➔ #H2A8B (H16)

read:var 1238 ➔ #H00002A8B (H32)

read:var 1238 ➔ #H0000000000002A8B (H64)

2.6.1.10 BINARY NUMERIC RESPONSE

Acutrol returns Binary data to the controller if the variable has been configured for B16, BH32, B64 output using the Configure:Variable command. The number of characters returned is 16, 32, 0r 64 for the respective formats.

Figure 10: Hexidecimal Response Data

Examples of data output for various word widths.

read:var 1238 ➔ #B0101010011110000 (B16)

read:var 1238 ➔ #B00000000000000000101010011110000 (B32)

2.6.2 VARIABLE NUMERIC FORMATS

The Acutrol control system supports all of the data formats listed in Section 2.8.1. This section needs to be revised according with IEEE guidelines on input data.

In this and most systems, 32-bit integers are adequate to represent the dynamic range of physical processes. For example, position data is typically scaled so that the full range is 1 revolution and the resulting resolution is still fine enough to measure sub-arc-second displacements. {I'd delete this, I'm not sure in the world today that 32 bits is enough, many optical encoder boards have 48 or higher bit accumulators. DELETE THIS SECTION}

2.6.2.1 BINARY DATA FORMAT

{This section needs to be updated for :CONF:VAR and for data types defined in Section 2.8.1}

The data type for state variables associated with the ACP (Axis Control Processor) are all 32-bit two's complement binary numbers (signed double words). Applications that require real-time data control and/or measurement should communicate over the IEEE-488 bus in binary using a block format. This interface provides commands to transfer blocks of related variables in binary format optimizing the performance of the interface. For example, the command VECTor sends a state demand vector consisting of position, rate, and acceleration for an axis as a block. {Nuke references to IEEE-488}

Other data formats and protocols have significant overhead resulting from messages that are typically verbose and require additional data type conversions and scaling.

Binary data may be transmitted over the interface using the <Definite Length Arbitrary Block Response Data> format described in Section 2.8.1.6. Binary data transmission (8-bit bytes, including the extended ASCII codes) begins as a byte serial stream after a block header is detected. This header begins with a "#" and is followed by at least one numeric character. A "0" (ASCII character) indicates that the block is an <Indefinite Length Arbitrary Block Response> and the series of data will be terminated with an <END>. If the value is other than "0", then the numeric value of the character indicates the number of specifier digits that immediately follow. The specifier digits indicate the number of bytes that follow in the data block. {Again, let's nuke Indefinite Length data for all but the IEEE-488}

The following examples illustrate sending four data bytes.

EXAMPLE: Arbitrary length block. In this example the "#" indicates that binary data is coming. The bytes that follow "0" until the <END> are binary data. {Again, let's nuke Indefinite Length data}

#0<lsb><byte><byte><msb><END>

EXAMPLE: A four byte block. In this example the "#" indicates that binary data is coming. The length of the binary data is determined by the next "1" byte which is a "4."

#14<lsb><byte><byte><msb><END>

EXAMPLE: An alternate four byte block. In this example the "#" indicates that binary data is coming. The length of the binary data is determined by the next "2" bytes which are "04."

#204<lsb><byte><byte><msb><END>

The largest block number supported for command data (download to Acutrol) is #299. The largest response (upload to the CIC) data specifier is #3250. {Is this an IEEE-488 limitation? Certainly not for TCP/IP or RS-232.}

{ACTION: Larry to review and determine actual limitations. We want to maximize for the datalogging module}

For additional information on the binary format see Appendix I. This appendix contains information pertaining to the scaling of variables and performance considerations. {This appendix doesn't exist, it's Section 6}

2.6.2.2 DECIMAL DATA FORMAT

Variables transmitted over the IEEE-488 interface as decimal numbers are sent as ASCII strings and represent scaled values with implied engineering units. The associated scale factor is defined for system variables when the system initially loads and is used to convert decimal values to and from binary numbers.

The scale factors are applied to variables transmitted over the IEEE-488 bus as ASCII decimal numbers in the same manner that the scale factors are applied to variables before being displayed on the Operator Interface Panel.

The decimal format is more readable than binary, and is conveniently used with input/output statements of higher level languages such as "C" or "BASIC." The penalty of course is significantly slower data rates.

The decimal data format used is defined as Numeric Response Data 2 or <NR2>, which is floating-point with an explicit decimal point and no exponent.

EXAMPLE:

Value	Full-Scale Range	Equivalent Resolution Range (bits)
270.00001	**360.0**	**25**
-2500.0001	**10000.0**	**27**
0.0001509913	**0.1**	**30**
2355009.1	**10000000.0**	**27**

In these examples, the dynamic range of values spans about 50 bits. These numbers in fact, represent different physical variables. However, with their respective scale factors applied, are all within the range of 32-bit numbers.

This approach has been selected over full floating-point numbers so that the entire 32-bit dynamic range can be applied to the mantissa. IEEE single precision floating-point provides only 24 bits for the mantissa and 8 bits for the exponent. In high precision systems, some of the variables require more than 24 bits to represent them with full accuracy.

For additional information on the Decimal Data Format, see Appendix I. {Section 6?} This appendix contains information pertaining to the scaling of variables and performance considerations.

2.6.3 NUMERIC COMMAND ARGUMENTS

Numbers that are used as command arguments are stored as integer values and are always communicated as a subset of <NR1> Numeric Data (integer values with an implied decimal

point). These numbers are typically used to select an axis or an option. They are transmitted as ASCII characters and never contain a decimal point or a negative sign. See section 2.6.1.1 Numeric Command Argument.

EXAMPLE: :READ:POSition *2*<END>

2.6.4 ALPHA-NUMERIC STRING DATA

Non-numeric data may be sent as a command argument such as when displaying a prompt message to the Operator Display Panel. The text argument is delimited with double quotes such that the length of the string need not be specified. A double quote may be embedded in the string by preceding it with a backslash.

	Data sent to Acutrol…	…is interpreted as this data.
EXAMPLE:	"This is a data string."	This is a data string.
	"This is "not" allowed."	<ERROR>
	"But \"this\" is allowed."	But “this” is allowed.

2.7 ENUMERATED TYPES

Enumerated types are used in commands when a command argument may only take one of a selected set of values. For example: the enable argument of the :**Configure:DemandSum** command may only have the values {Enable | Disable}. When a command type is specified as [enum], followed by a set of values, the command will accept any of those values. Values will be recognized once a unique number of characters are entered. In the example above “e” or “*E*” is sufficient to uniquely identify “Enable”.

3 COMMON COMMANDS

3.1 *CLS

Clear Status Command.

This command clears the status register and its associated status data structures that are summarized in the Status Byte (such as the Event Status Register). This command also clears all status-related queues except the Output Queue.

3.2 *ESE[?]

Standard Event Status Enable Command and Query.

This command sets the Standard Event Status Enable Register bits. The data is formatted as <NRf> in the range of 0 to 255. Acutrol will round the number to an integer. The binary equivalent of this decimal number represents the bits to be set in the Standard Event Status Enable Register.

COMMAND: *ESE <Standard Event Status Enable Register Data><END>

EXAMPLE:
*ESE 0<END> Clear all bits.
*ESE 16<END> Set bit 4 (DIO5).
*ESE 255<END> Set all bits.

The query command reads the contents of the Standard Event Status Enable Register. Acutrol will return this value as <NR1> data in the range of 0 to 255.

QUERY: *ESE?<END>

➔<Standard Event Status Enable Register Data><END>

EXAMPLE:
*ESE?<END>
➔0<END> No bits are set.
*ESE?<END>
➔255<END> All bits are set.

3.3 *ESR?

Event Status Register Query.

This command reads the contents of the Standard Event Status Register. Acutrol will return <NR1> data in the range of 0 to 255. The binary equivalent of this decimal number represents the bits set in the Event Status Register. Reading this register clears it.

QUERY: ***ESR?<END>**

➔<Event Status Register Data><END>

EXAMPLE: *ESR?<END>
➔0<END> No bits set.

*ESR?<END>
➔16<END> Bit 4 set (DIO5).

3.4 *IDN?

Identification Query.

This command causes Acutrol to identify itself. The response to this command is formatted as <Arbitrary ASCII Response Data>.

QUERY: ***IDN?<END>**

➔<manufacturer>,<model>,<serial number>,<software level><END>

EXAMPLE: *IDN?<END>
➔Acutronic R&D USA,Acutrol3000,0,A8800-001-01.000-a<END>

The length of the returned string will not exceed 72 characters including the commas. The <serial number> and <software level> fields will be different than shown above.

3.5 *IST?

Individual Status Query.

(This is a GPIB command only and is not currently implemented.)

This command causes Acutrol to respond with the current state of the *ist* local message. The *ist* is the individual status that Acutrol will send when Parallel Polled. The state of this bit is determined by the Parallel Poll Enable Register and corresponding status information.

The response to this command is formatted as <NR1> data. A value of ASCII 0 (hex 30) will be returned if *ist* is false and ASCII 1 (hex 31) if *ist* is true. This allows a controller to read the Acutrol response to a Parallel Poll without performing the Parallel Poll.

QUERY: ***IST?<END>**

➔<ist local message><END>

EXAMPLE: *IST?<END>

➔ 0<END> Acutrol ist is false.

*IST?<END>

➔1<END> Acutrol ist is true

.

3.6 *OPC[?]

Operation Complete and Operation Complete Query.

(This command is not implemented at this time.)

This command causes Acutrol to set the LSB (DIO1) in the Standard Event Status Register when all pending operations are complete.

COMMAND: ***OPC<END>**

Implement this by AND'ing the Profile Complete Bits together for all the axes in a system. Define all commands as sequential for Phase 1, Defer overlapped commands until version 2.

The query causes Acutrol to output an ASCII 1 (hex 31) when all pending operations are complete.

QUERY: ***OPC?<END>**

➔1<END>

Implement this by AND'ing the Profile Complete Bits together for all the axes in a system. Needs a method to abort for all interfaces. Define all commands as sequential for Phase 1, Defer overlapped commands until version 2.

3.7 *PRE[?]

Parallel Poll Enable Register Enable Command and Query.

(This is a GPIB command only and is not implemented at this time).

This command sets the bits in Acutrol's Parallel Poll Enable Register. These bits determine what conditions set the *ist*. This command is followed by <NRf> format data. The value of this number must round to a value in the range 0 to 65,535. This number, when converted to binary, represents the bit to be set in the register.

COMMAND: ***PRE <Parallel Poll Enable Register Data><END>**

EXAMPLE: *PRE 0<END> No condition will set ist.

*PRE 65535.0<END> All conditions will set ist.

The query command returns the current value of the Parallel Poll Enable Register, as set by the *PRE command, as <NR1> formatted data.

QUERY: ***PRE?<END>**

➔<Parallel Poll Enable Register Data><END>

EXAMPLE: *PRE?<END>

➔0<END> No conditions will set ist.

*PRE?<END>

➔256<END> One condition (bit 8) will set ist.

3.8 *RST

The Reset command is used to put the controller in a defined state.

(This command is not implemented at this time.)

Acutrol performs the following actions when this command is received:

- Sets all axes to Off (Zero Rate) mode
- Sets all demands for all axes to 0 (or is this the center of the full-scale range)
- Sets all limits to the default values (as defined in the XML file)
- Stops the execution of all user macro programs
- Aborts any data logging in progress
- Clears any previous *OPC commands
- Clears any previous *OPC? Queries
- Open Interlocks via soft abort mode
- Free all Acquisition Channels
- Reinitialize the system from the XML as a power cycle

This command will not affect the following: (Update this)

- The Acutrol output queue
- The execution of any system macro program
- The state of the IEEE-488 interface or the IEEE-488 address
- The Service Request Enable Register
- The Standard Event Status Enable Register
- The power-on flag
- Calibration data

COMMAND: ***RST<END>**

3.9 *SRE[?]

Service Request Enable Command.

(This is a GPIB command only.)

This command sets bits in the Service Request Enable Register, which determines what bits in the Status Byte will cause a Service Request from Acutrol. The data sent with this command is formatted as <NRf>. Acutrol will round this number to an integer. The binary equivalent of this number determines the bits that will be set in the Service Request Enable Register.

COMMAND: ***SRE <Service Request Enable Register Data><END>**

EXAMPLE: *SRE 0<END> Clear all bits.

*SRE 32<END> Set bit 5 (DIO6).

The query command reads the contents of the Service Request Enable Register. Acutrol will return this formatted as <NR1> data in the range 0 to 63 or 128 to 191 because bit 6 (RQS) cannot be set.

QUERY: ***SRE?<END>**

<Service Request Enable Register Data><END>

EXAMPLE: *SRE?<END>

→0<END> Not bits set.

*SRE?<END>

→ 32<END> Bit 5 (DIO6) set.

Note: The Event Status Register Definitions must include all appropriate items from the following ACT2000 table after which this table can be removed.

Table 1: SRQ Enable Mask and Event Status Bits

Bit	Description
15–08	Reserved for future use.
07	:UTILity:TOUCh command completed.
06	Reserved.
05	Axis timed-out. After a mode or servo operation, this bit indicates that the axis did not respond as commanded.
04	After the Local to Remote Mode change this bit indicates a fault during the mode transition. After the :CONFig:OVARiable command this bit indicates that no data selects were available on the requested axis. There is a maximum of 5 free data selects per axis while in Remote Mode.
03	All axis profiles complete. Set if all of the Command Processor profiles are complete. Will not be signaled again until at least one of the axes cycles from complete to not complete to complete.
02	Bad device dependent message received. Signals that the message had one of the following errors: Axis number out of range. Data not the correct size. Binary data not in binary format. Not enough binary data. OVARiable number out of range. **Note:** unknown commands are ignored and do not generate an SRQ.
01	Reserved
00	MIS servo fault. Only set if an axis was previously servoed or an attempt to servo an axis failed.

3.10 *STB?

Status Byte Query.

This command reads the Status Byte with the MSS (Master Summary Status) bit. Acutrol will return <NR1> formatted data in the range of 0 to 255. The binary equivalent of this number determines the bits that are set in the Status Byte. Note that bit 6 (DIO7) represents MSS and not RQS (Request Service).

Standard Status Byte Register (STB) is defined by the IEEE Std 488.2 standard as follows:

Bit 7 (USER): Device specific summary bit (defined below).

Bit 5 (RQS/MSS): "Master Summary Status" indicates that at least one enabled event has occurred.

Bit 5 (ESB): "Event Status Bit" indicates that one or more of the enabled events Standard Events has occurred since the register was last read or cleared.

Bit 4 (MAV): "Message Available" indicates that a message is available in the Output Queue.

Bits 0-3 (USER): Device specific summary status bits (defined below).

The ACUTROL 3000 uses bits 0-3 and bit 7 of the STB as follows:

Bit 7 (EQ): Summary bit for Error Message Queue.

Bit 3 (RI): Summary bit for Remote Interfaces

Bit 2 (SUP): Summary bit for Supervisory Processes

Bit 1 (ACP): Summary bit for ACP Processes

Bit 0 (SI): Summary bit for System Interlocks

Note: Reading the STB will clear all of the bits.

QUERY: ***STB?<END>**

➔<Status Byte with MSS><END>

EXAMPLE: *STB?<END>
➔65<END> Bit 6 (MSS, DIO7) and bit 0 (DIO1) set.

*STB?<END>
➔2<END> Bit 1 (DIO2) set.

3.11 *TRG

Trigger Command.

This command performs the same function as GET (Group Execute Trigger) on a device-by-device basis. This will initiate some previously defined command.

Commands supported are:

1. Datalogging: This will initiate the data logging process only if the :ACQUIRE:ARM:WAIT command has been previously set.
2. Playback: This will initiate the data logging process only if the :Playback:ARM:Wait command has been previously sent.

COMMAND: ***TRG<END>**

3.12 *TST?

Self-Test Query.

(This command is not implemented at this time.)

This command causes Acutrol to report on the results of its POST (Power-On Self-Test). The POST performed by Acutrol is independent of the POST performed by the BIOS in Acutrol.

The response to this command is formatted as <NR1> data in the range of -32767 to 32767. A value of 0 indicates that all self-tests completed without detecting errors. Non-zero values will indicate only the first detected error as described in the following table.

{This needs some work. What if multiple faults occur? How about a hexadecimal value with each bit representing a test?}

{Change so that 0 = no error, non-zero = error, read status words for clarification)

QUERY: ***TST?<END>**

➔<POST results><END>

EXAMPLE:
*TST?<END>
➔0<END> No errors were detected.

*TST?<END>
➔7<END> Supervisor (Axis 0) failed test 7.

*TST?<END>
➔1012<END> Axis 1 failed test 12.

Table 2: Self-Test Query Response Format

POST Character	1000's	100's thru 1's
Failing Axis and Test	Axis Number (Supervisor is 0)	Number of the first detected failing test

Table 3: Supervisor Self-Tests

Supervisor Test	Test Description
1	TBD

Table 4: Axis (ACP) Self-Tests

Axis Test	Test Description
1	TBD

3.13 *WAI

Wait to Continue.

(This command is not implemented at this time.)

This command causes Acutrol to stop accepting commands from this interface until all overlapped commands and queries have completed.

COMMAND: *WAI<END>

Needs a method to abort for all interfaces.

Define all commands as sequential for Phase 1, Defer overlapped commands until version 2.

4 ACUTROL3000 COMMANDS

The commands defined in this section constitute the family of instructions, which are used to communicate with the Acutrol3000 motion controller. The complete set of commands is refered to as the Acutronic Command Language (ACL) and is used as the communication protocol for all of the supported non-real-time interfaces. The ACL is used to accomplish the following functions:

- Control the motion states of a system, including servo modes and motion commands.
- Monitor the state of the controller, including motion states/variables and status.
- Management of data logging of system variables.
- Configuration of controller features and system topologies.
- Calibration of system/transducer errors.

The Acutronic Command Language has many commands that have been grouped into subsystems for convenient access to commands of similar function (control, monitor and operational functions). The subsystem categories in the MAIN root are summarized below:

- **Acquire -** Provides set up of channels and control of the data logging functions.
- **Configure -** The Configure subsystem includes commands to assign specific variables to analog I/O, Encoder and Servo subsystems, and various other features that are unique to the controller.
- **Demand -** Defines motion state demands to the axis servo variables.
- **Interface** – Used to configure the various digital interfaces that are used by remote computers to communicate with Acutrol.
- **Interlock -** Sends axis commands to control the operational state of the system servos.
- **Limit -** Programs the operating limits of the motion states to provide a safe operating environment and smooth motion transitions.
- **Mode -** Selects the servo mode for each axis.
- **Platback** – Configures the Playback of pre-defines motion scenarios. This is a future feature planned for 1Q2006.
- **Query** – (No commands implemented at this time)
- **Read -** Selects variables to be measured and/or returns their values.
- **Status -** Provides motion controller subsystem operating status and related information.
- **System** – Used to configure or query system level features of the control system.
- **Utility -** The Utility commands facilitate communication and interaction with a human operator as well as other service commands.

The Acutrol Act 1000 series and Act 2000® support the device dependent messages listed in Table 5. The Acutrol3000 will also support these messages. In addition, the Acutrol3000 adds support for queries of these commands as listed in Table 6 and the new commands listed in Table 7.

When porting existing GPIB application from the Acutrol Act 1000 series and Act 2000 to the Acutrol3000, the user must change or remove any references to the messages listed in Table 8: Commands that Behave Differently. These messages do not function or exhibit different behavior than they did previously.

Table 5: Old Commands and Queries

:Configure:Analog	**:Interlock:Close**	**:Mode:ARate**	**:Status:Accumulator**
:Configure:Discrete	**:Interlock:Open**	**:Mode:Position**	**:Status:ECP**
:Configure:Ovariable	**:Interlock:Reset**	**:Mode:RRate**	**:Status:MIS**
:Configure:Window		**:Mode:Synth**	**:Status:Supervisor**
	:Limit:Acceleration	**:Mode:Track**	
:Demand:Acceleration	**:Limit:HiPosition**		**:Utility:Accumulator**
:Demand:Oscillator	**:Limit:LoPosition**	**:Read:Acceleration**	**:Utility:Beep**
:Demand:Position	**:Limit:Rate**	**:Read:Fdelay**	**:Utility:Format[?]**
:Demand:Rate	**:Limit:Vtrip**	**:Read:Ovariable**	**:Utility:Lockout**
:Demand:ShortVect		**:Read:Position**	**:Utility:Prompt**
:Demand:Vector		**:Read:Rate**	**:Utility:SRQ[?]**
		:Read:ShortVect	**:Utility:Touch**
		:Read:TimeClock	
		:Read:Vector	

Table 6: Queries Added to Old Commands

:Configure:Analog?	**:Interlock?**	**:Mode?**	
:Configure:Discrete?			
:Configure:Ovariable?			
	:Limit:Acceleration?		
:Demand:Acceleration?	**:Limit:HiPosition?**		**:Utility:Lockout?**
:Demand:Oscillator?	**:Limit:LoPosition?**		
:Demand:Position?	**:Limit:Rate?**		
:Demand:Rate?	**:Limit:Vtrip?**		
:Demand:ShortVect?			
:Demand:Vector?			

Table 7: New Commands Added

***IDN?**	**:Acquire:Activate[?]**	**:Configure:RealTime:Translate[?]**	**:Interlock:Logic[?]**
***RST**	**:Acquire:Channel[?]**	**:Configure:Restore[?]**	**:Limit:Absolute[?]**
***TST?**	**:Acquire:Free[?]**	**:Configure:Restore:Group[?]**	**:Limit:BEmf[?]**
***OPC[?]**	**:Acquire:Read[?](obsolete)**	**:Configure:Save[?]**	**:Limit:Global[?]**
***WAI**	**:Acquire:Sample[?]**	**:Configure:Save:FileVersion[?]**	**:Limit:Jerk[?]**
***IST?**	**:Acquire:Save**	**:Configure:Save:Group[?]**	**:Mode[?]**
***PRE[?]**	**:Acquire:Status[?]**	**:Configure:Scaling[?]**	**:Mode:Off**
***CLS**	**:Acquire:Trigger[?]**	**:Configure:RealTime[?]**	**:Mode:Home(future)**
***ESE[?]**	**:Configure:AIM[?]**	**:Configure:RealTime:BlockRecord[?]**	**:Mode:Arate**
***ESR?**	**:Configure:Axis[?]**	**:Configure:RealTime:Monitor[?]**	**:Mode:Rate (deviation)**
***SRE[?]**	**:Configure:Conditioner[?]**	**:Configure:RealTime:Setup[?]**	**:Playback:Activate[?]**
***STB?**	**:Configure:Correlation[?]**	**:Configure:RealTime:Tracker[?]**	**:Playback:Configure[?]**
***TRG**	**:Configure:DemandSum[?]**	**:Configure:Summer[?]**	**:Playback:Data[?]**
	:Configure:DigitalIO[?]	**:Configure:System:Restore[?]**	**:Playback:List[?]**
	:Configure:Event[?]	**:Configure:System:Save[?]**	**:Playback:Restore[?]**
	:Configure:FbkChannel[?]	**:Configure:Variable[?]**	**:Playback:Save[?]**
	:Configure:Filter[?]		**:Playback:Trigger[?]**
	:Configure:Filter:Alternate[?]	**:Demand:DeltaPosition[?]**	**:Read:Jerk**
	:Configure:Filter:Select[?]	**:Demand:InputVariable[?]**	**:Read:LogData[?]**
	:Configure:GPIO[?]	**:Demand:Oscillator[?]**	**:Read:Variable[?]**
	:Configure:LUT[?]	**:Demand:WorldCoordinates[?]**	**:Read:World: Coordinates[?]**
	:Configure:Filter:Select[?]	**:Demand:Jerk[?]**	
	:Configure:GPIO[?]		**:Status:CSR?**
	:Config:Home[?](future)	**:Interface:IEEE488[?]**	**:Status:MSG?**
	:Configure:LUT[?]		**:Status:MSGClr?**
	:Configure:LUT:Fourier[?]	**:Interface:Parallel[?]**	**:Status:STB?**
	:Configure:LUT:List[?]	**:Interface:RS232[?]**	**:Status:TFR?**
	:Configure:LUT:Mode[?]	**:Interface:ScramNet:CSR[?]**	**:Utility:RemoteLockout[?]**
	:Configure:LUT:Series [?]	**:Interface:ScramNet:Mode[?]**	**:Utility:Macro?**
	:Configure:Model[?]	**:Interface:ScramNet:Reset[?]**	**:Utility:Macro:Close**
	:Configure:MotionStates[?]	**:Interface:VMIC[?]**	**:Utility:Macro:Execute**
	:Configure:Motor[?]	**:Interface:VMIC:CSR[?]**	**:Utility:Macro:Load**
	:Configure:Optical[?]	**:Interface:VMIC:Reset[?]**	**:Utility:Macro:Open**
	:Configure:Oscillator[?]	**:Interface:Reflectivemem:Setup[?]**	**:Utility:Macro:Remove**
	:Configure:Plant[?]	**:Interlock:Configure[?]**	**:Utility:Macro:Stop**
	:Configure:Ratiometric[?]	**:Interlock:Control[?]**	**:Utility:Macro:Upload**

Table 8: Commands that Behave Differently

:Configure:Ovariable		**:Status:Accumulator**	**:Utility:Accumulator**
		:Status:ECP	**:Utility:Beep**
:Interlock:Reset	**:Read:Fdelay**	**:Status:MIS**	**:Utility:Prompt**
		:Status:Supervisor	**:Utility:SRQ[?]**
			:Utility:Touch

4.1 ACQUIRE

The **Acquire** Subsystem provides commands to define and enable the data logging features of the Acutrol control system. There are several steps associated with setting up data logging. These steps are accomplished using the **Channel**, **Free**, **Sample**, **Trigger**, and **Activate** subcommands.

- The data to be logged is specified by assigning system variable numbers to data logging channels using the **Acquire:Channel** command.
- The **Acquire:Channel** command is also used to specify the "set operation" for each channel if multiple data sets are to be collected and combined. The maximum number of channels that may be simultaneously recorded is defined by a system parameter, **MAX_ACQ_CHANNELS** and is set at the factory. It is currently set at 64.
- The **Acquire:Free** command is used to release previously assigned channels.
- The **Acquire:Sample** command sets the size of the data buffers and the time interval between samples. The total number of samples[1] is limited by the amount of memory allocated for data logging (set at the factory) and the number of active channels. Reducing the number of channels will make more memory available for samples, and vice versa. Similarly, the number of active channels is limited by the number of samples specified. Reducing the number of samples will make more channels available. The time interval between samples is specified as an integer number of Acutrol process frames. All parameter limits are set at the factory except as described above.
- **:ACQUIRE:STATUS?** reports the current datalogging state and the value of the current sample and set.
- Also, using the **Acquire:Sample** command, more than one data set may be requested. In this case, the sets are combined into each buffer using any of several "set operations" including **Averaging, Minimum Value, Maximum Value, and Raw.** If averaging is requested in the channel's set operation parameter, the stored value would be the sum of the previously stored value and the newly sampled value. See the Channel subcommand for a description of all set operations. When the data is read, each sample is first be divided by the total number of samples to produce an average.
- **Starting the Data Log:** There are three commands that can be used to start data logging: **Acquire:Activate**, **ArmAcquire:**, **Activate Now**, and ***TRG**. **ARM** causes data logging to begin when the criteria given as parameters to the **Acquire:Trigger** command have been met. **NOW** and ***TRG** cause data logging to begin immediately.
- **:Acquire:Trigger** command - triggering may be set to occur when the value of the specified trigger channel's variable exceeds a trigger level setpoint while changing in the direction specified by the sign argument. In this mode, each data set is triggered independently. A trigger level increment delta can be specified to adjust the trigger level before the beginning of each data set.
- **Stopping the data log:** The following commands can be used to stop data logging: **Acquire:Activate STOP**, **Acquire:Activate ABORT**, or completing the acquisition of the number of sample sets specified in the Acquire:Sample command. **Acquire:Activate STOP** will cause data logging to cease at the end of the current data set. **Acquire:Activate ABORT** will cause data logging to cease immediately.

[1] Including pre-triggered samples.

Some typical command sequences are shown in the tables below:

Take a single set of data samples for two variables upon command.

Commands	Description
Acquire:Channel 1, 1008	Record variable 1008 on channel one.
Acquire:Channel 2, 1009	Record variable 1009 on channel two.
Acquire:Sample 5, 10000, 1	Sample every fifth ACP frame; stop after one set of 10000 samples has been acquired.
Acquire:Activate NOW	Start sampling.

Take multiple data sets starting with a data sensitive trigger event.

Commands	Description
Acquire:Channel 1, 1008, AVERAGE	Record variable 1008 on channel one. Add new values to previously stored values.
Acquire:Channel 2, 1009, RAW	Record variable 1009 on channel two. Replace data set values with new values as they are sampled.
Acquire:Sample 1, 10000, 10	Record data for every ACP frame until 10 sets of 10000 samples each have been acquired.
Acquire:Trigger 2, 0.0, +, 0.01	Begin the first data set collection when channel two's variable crosses zero going positive. For subsequent data sets, increase the trigger level by 0.01 for each set. In other words, the last data set in this example will begin when variable 1009 (Ch 2) changes from a value below 0.09 to one above 0.09.
Acquire:Activate ARM	Begin trigger condition monitoring.

Take data continuously until 'n' samples after a data sensitive trigger event occurs.

Commands	Description
Acquire:Channel 1, 1008, REPLACE	Record variable 1008 on channel one. Replace data set values with new values as they are sampled.
Acquire:Channel 2, 1009, REPLACE	Record variable 1009 on channel two. Replace data set values with new values as they are sampled.
Acquire:Sample 1, 10000, n	Record data for every ACP frame until the trigger event occurs and then sample 'n' more times.
Acquire:Trigger 2, 0.0, -, STOP ***(This feature is not implemented at this time.)***	Initiate the stopping of data collection when channel two's variable crosses zero going negative. Note that the 'delta' parameter of Trigger has no meaning for this trigger mode.
Acquire:Activate WAIT	Begin data collection when the *TRG common command is received.

4.1.1 ACQUIRE:ACTIVATE

The Acquire:Active command specifies the sampling and trigger processes of the data logging system. This command has a single parameter that may take on any of six values. Query for this command is not defined and the active state of datalogging is obtained hy using the **Acquire:Status?** Query.

COMMAND: :Acquire:Activate {ARM | NOW | STOP | ABORT | RESET}

ARM - Commence logging data into a circular buffer whose maximum record size is determined by the **Acquire:Samples <sample>** argument. Upon detection of a trigger event, additional records will be logged corresponding to the number of samples specified minus the pre-trigger value defined in the **Acquire:Trigger <#pre-trig samples>** command.

NOW - Commence data logging immediately and stop after collecting the specified number of samples.

STOP - End data logging at the completion of the current data set.

ABORT - End data logging immediately.

RESET - End a previously started acquisition session. This command disables any further retrieval of logged data and enables the setting of parameters for a new data acquisition session.

EXAMPLE: :ACQ:ACT ARM

Errors:

1. No channels defined. (Does nothing, reports error to status byte as ILLEGAL COMMAND Bit)
2. Acquisition in process (except ABORT or STOP or OFF). (Restarts acquisition)

Notes:

4.1.2 ACQUIRE:CHANNEL[?]

The Acquire:Channel command is used to assign system variables to the sampling channels of the data logging system. See Acutronic manual TM-TBD, Acutrol 3000 Variable Definitions for a list of variable numbers.

COMMAND: **:Acquire:Channel <channel>,<variable>,[<set_oper>]**

- <**channel**> – The data log channel number. [0 .. MAX_ACQ_CHANNELS-1]
- <**variable**> – The number of the variable to record.
- <**set_oper**> - Defines how multiple sets of data are processed.
  - **AVERAGE** – Use this option to take multiple data sets and record the average of each sample value.
  - **MIN** - Use this option to take multiple data sets and record the minimum value (on an absolute value basis) of each sample.
  - **MAX** - Use this option to take multiple data sets and record the maximum value (on an absolute value basis) of each sample.
  - **RAW (DEFAULT)** – use this option to have each data set overwrite the previous value. If no set operation is specified, RAW is used.

EXAMPLE: :ACQ:CHAN 1,1023

:A:C 2,2055,AVERAGE;C 3,3055

Query of the **Acquire:Channel** command will report the variable number of the channel specified and the current set operation for that channel.

QUERY: **:Acquire:Channel? <channel>**

→ <variable>,<set_oper>

EXAMPLE: :Acq:Ch? 2;Ch? 2

→ 2055,Average; 3055,Raw

Querying a channel that has not been defined will return a <variable> of 0 and a <set_oper> of Raw.

EXAMPLE: :A:C? 7;

→ 0,Raw

Querying a channel that was defined but has since been freed, will indicate the "zero variable" (X118) and the <set_oper> will be unchanged.

EXAMPLE: :A:C? 7;

→ 1118, Average

To determine if a channel is active or free, use the :Acq:Free? command.

Errors:

1. Invalid channel number – not in range [1,NUM_ACQUIRE_CHANNELS]. Sets PARAMETER OUT OF RANGE bit in Status Byte.
2. Variable number out of range. Sets PARAMETER OUT OF RANGE bit in Status Byte
3. Acquisition in Process. Sets ILLEGAL CMD bit in Status Byte

Notes:

4.1.3 ACQUIRE:FREE

The **Acquire:Free** command is used to remove one or more channels from the list of channels being used to log data. If the freed channel has been specified as the trigger channel, that designation is deleted also.

COMMAND: **:Acquire:Free {<channel> [, <channel> […]] | ALL}**

- **<channel>** – A channel number to free.
  ALL – use this option to release all previously assigned channels.

EXAMPLE: :ACQ:FREE 1,2,4
:ACQ:FREE ALL

Querying the :ACQ:FREE command will return a list of unassigned channels:

QUERY: **:Acquire:Free?**
→ <channel> [, <channel> […]]

EXAMPLE: : Acq:FREE?

→5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32

In this example, channels 1-4 are active.

Errors:

1. Invalid channel number. Set PARAMETER OUT OF RANGE Error Bit in Status Byte.
2. Query. Sets the ILLEGAL CMD

Notes:

1. DCL or *RST will free all acquisition channels.

4.1.4 ACQUIRE:READ?

THIS COMMAND IS NOT CURRENTLY IMPLEMENTED BUT WILL BE ADDED TO SUPPORT DATA LOGGING USING THE IEEE-488 INTERFACE PROTOCOL.

This command is used to read the data logged by the ACQuire Subsystem. The <channel> argument specifies the channel of logged data to read. Acutrol responds with <DEFINITE LENGTH ARBITRARY BLOCK RESPONSE DATA> (see TK-INSERT REFERENCEHERE).

The number of data points that may be read at one time may be limited by a system parameter (set at the factory), <num_pts>. The interface computer (CIC) continues to issue :Acquire:read commands until all data is collected. In this case the logged data will be broken up across multiple reads until all data is read. Data are not destroyed when read and may be re-read until the log channel is freed, or a new set of data is acquired. Data can not be read while data is being logged.

Note: Issuing a command without the <index> field will return the next <num_pts> of data.

COMMAND: **:Acquire:Read? <channel>,<index>**

→<index> #dbbXXXXXXXXXXXX

- **<channel>** – The data log channel number to read.
- **<index>** – The sample index to start reading data. If <index> is not specified, then the starting sample is the last reported sample plus 1.
- **<#dbbXXXXXXXXXXXX >** – The data in <DEFINITE LENGTH ARBITRARY BLOCK RESPONSE DATA> format.. I.e., the pound sign (#), a non-zero digit, d, specifying the number of digits to follow, d digits representing the number of data bytes, followed by the data bytes.

  For example, if samples is 1024, the following would be returned:

  #48096xxx…xxx

  where xxx…xxx represents 8096 bytes of data. The data returned is in double precision format (8 bytes per data log point), ordered in little endian format (lsb to msb).

EXAMPLE: :Acquire:read? 3, 0

→ 0,#44096xxx…xxx

Errors:

1. Invalid <channel>,<index>,<num_pts>. Set the PARAMETER OUT OF RANGE bit.

4.1.5 ACQUIRE:SAMPLE[?]

The **Acquire:Sample** command is used to specify the basis for sampling data. The three arguments define the sampling **<interval>**, the number of **<samples>**, and the number of data **<sets>**. An auto save can be enabled to save the data stored in RAM to file on the drive of the RT computer immediately after the data log has completed.

COMMAND: **:Acquire:Sample <interval>,<samples>,<sets>[,<autosave>]**

- **<interval>** – The number of ACP frames between samples. (E.g., 1 means sample every frame, 2 means sample every other frame, 3 every 3rd frame, etc.)
- **<samples>** – The number of samples to record. If pretriggering was specified in the :acquire:trigger command, then the total number of samples saved will be <samples> + <pre-trig samples>
- **<sets>** – The number of data sets to acquire.
- **<autosave>** [True|False] – If <autosave> is **True** then logged data will be saved to the current default file after data logging is complete.

EXAMPLES: :ACQ:SAMP 10,1024,1, False

:ACQ:SAMP 1, 45000, 1

QUERY: **:Acquire:Sample?**

→ <interval>,<samples>,<sets>,<autosave>

EXAMPLE: :ACQ:SAMP?

→ 1, 45000, 1, False

Errors:

1. Interval out of range – set PARAMETER OUT OF RANGE bit in Status Byte.
2. Samples out of range – set PARAMETER OUT OF RANGE bit in status byte.
3. Count out of range – set PARAMETER OUT OF RANGE bit in status byte.
4. Acquisition in Process. Sets ILLEGAL CMD bit in Status Byte

Notes:

Except when pre-triggering, the logging process stops automatically when all of the requested data is sampled. In all cases, it may also be stopped by using the STOP or ABORT or RESET parameters of the **:Acquire:Activate** command.

4.1.6 ACQUIRE:SAVE

This command allows acquisition data, which is stored in RAM, to be saved to the local hard drive in the LynxOS control computer.

COMMAND: **:Acquire:Save**

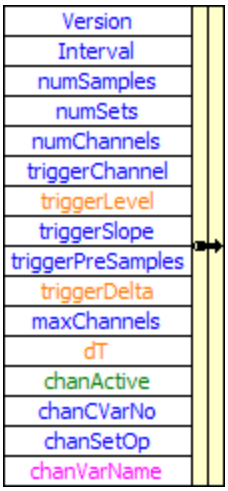
The current file specification for the destination file cannot be changed; therefore, the data log file must be manually copied to another location and saved/renamed; otherwise, the data will be overwritten the next time logged data is saved. In the future, this command may be expanded to include a file specification to define the default file directory and name.

Currently, data is transferred to the GUI or another user computer using an FTP client and the TCP/IP Ethernet connection.

The logged data is saved in the following file on the Acutrol RT Flash drive:

root/mnt/flash/datalog/LOG_A3KDATA.dlog

The data file contains a header, which defines the specifics of the data logging session.



The header and each sample record are sequentially stored in the file in double float binary format.

Note: The :Acquire:Save? (query) command in is not defined and will return an error. The Query command will be activated if the file spec feature is implemented.

4.1.7 ACQUIRE:STATUS?

Returns data logging status.

COMMAND: :Acquire:Status? <state>, <sampleCount>, <setCount>

- **<state>** [enum] {INIT | READY | ARMED | ON | DONE | ERROR} returns current state of the data logging system. The six possible return values are:

  INIT -Data logged is initializing.

  READY – Data logging is configured and ready. No data is being logged and triggering has not been enabled.

  ARMED - Waiting for trigger to occur.

  ON -Data is currently being logged.

  DONE – Logged data is ready for retrieval.

  ERROR – A data logging error occurred.

- **<sampleCount>** [NR1] returns the current sample count {0 .. MAX_ACQ_COUNT}.

- **< setCount >** – [NR1] return current set count { 0.. N}.

4.1.8 ACQUIRE:TRIGGER[?]

The Acquire:Trigger command is used to specify the basis of the start/stop event, which controls the data logging process.

COMMAND: **:Acquire:Trigger <channel>,<level>,<slope>[,<#pre-trig samples>[,<delta>]]**

- < **channel** > – The trigger channel number.
- < **level** > – The trigger threshold value. <level> is in the same units as the variable associated with the trigger channel. This is combined with the <sign> parameter to determine when triggering will occur.
- <**slope**> – The trigger slope, '+' or '-'. Used to specify the direction that the trigger channel's value should be going when triggering occurs.
- <**pre-trig samples**> - the number of samples to capture before the trigger event occurs.
- <**delta**> - If not pre-triggering (i.e., pre-trig samples equals 0), <delta> is the amount that <level> should be changed at the end of each data set. In other words, the first data set is triggered when the channel variable's value passes through <level>, the second set when the value passes through <level> + <delta>, …, the nth set when the value passes through <level> + n*<delta>.

EXAMPLE: :ACQ:TRIG 1,10.0,+,1.0

:A:T 2,100,-

:A:T 2,,+

A TRIGger query reports the current values of the trigger setup parameters.

QUERY: :Acquire:Trigger?

➔ **<channel>,<level>,<sign>,<#pre-trig samples>,<delta>**

EXAMPLE: :A:TRIG?

➔ 2,100.0000,+,0,0.00000

:A:TRIG?

➔ 1,10.00000,+,1,-1.00000

Errors:

1. Invalid channel number. Sets the PARAMETER OUT OF RANGE bit in the Status Byte.
2. Acquisition in Process – Set the ILLEGAL CMD bit in the Status Byte
3. Level value out of range of variable – Set the PARAMETER OUT OF RANGE bit in the Status Byte.

4. What happens if delta moves to out of range? Or wraps on position? The trigger level should be constrained to the pre-defined limits of the trigger variable.

Notes:

Some parameter "errors" that are not specifically flagged but may cause problems for the user are:

1. The value <level> is inconsistent with the selected variable. i.e., positions outside of [-180,180] on some systems.
2. A negative <sign> supplied for a non-decreasing variable.
3. A <delta> that will drive the level out of range before the completion of the logging of all data sets.
4. Channel may not be assigned to a variable.

4.2 CONFIGURE

The **Configure** commands in this section are used to customize most of the hardware and software functions of the controller. These functions are broken down into blocks that are programmatically tailored and connected together thus defining the computational data flow and consequently the architecture of the Acutrol3000 controller. In general specifying Acutrol System Variables configures inputs to a functional block. Parameters define the operational mode, scaling, and calibration of the block. The output of a block may directly affect hardware or data structures; or, it may simply update associated system variables making them available for subsequent block operations. Configuring a block is accomplished using ACL commands to write to the ports and registers that define the operating parameters for the associated functions. For a definition of the Acutrol3000 variables refer to Acutronic manual Variable Definitions TM-9391. Also, configuration parameters and variables are further explained in the various integration manuals and checkout procedures.

4.2.1 CONFIGURE:AIM[?]

This command is used to define the discrete-time update rate for the control processes and to specify which Axis Interface Module is dedicated as the source for distributing the Frame and 10kHz synchronization signals on the AIM Sync connector. This Supervisory command is used to configure all axes/AIMs in the system. Axis 1 is the default master.

COMMAND: **:Configure:AIM <Update_Freq>,<master frame axis>,<master 10k axis>,**
<sw cap filter freq>,<clockAdjust>,<aim delay>

- **<Update_Freq>** [Enum] {10000 | 5000 | 3333 | 2500 | 2000 | 1667 | 1429 | 1250 | 1111 | 1000 | 909 | 833 | 769 | 714 | 667 | 625} Defines the nearest frequency based on the integer number of 10kHz cycles that make up the sample Frame period. E.g., <frame period> = 2000 implies a 500µs period, which corresponds to 5 cycles of the 10 kHz. The enumerated arguments must be entered exactly as they appear above.
- **<master frame axis>** [NR1] specifies the axis that is enabled to output the Frame Sync signal on the AIM Sync connector. Defaults to Axis 1
- **<master 10k axis>** [NR1] specifies the axis that is enabled to output the 10kHz Sync signal on the AIM Sync connector. Defaults to Axis 1.
- **<sw cap filter freq>** [Enum] {0 | 125 | 250 | 500 | 625 | 1000 | 1250 | 2500 | 5000} specifies the break frequency in Hz for the switched capacitor filters on the four AIM analog inputs. **<sw cap filter freq>** Defaults to 2500 Hz. The enumerated arguments must be entered exactly as they appear above.
- **<clockAdjust>** [NR2] {ppm} AIM oscillator user offset.
- **<aim_delay>** [NR2] {µseconds} User defined programmable delay that is used to adjust the reference point for time skew correction of the AIM clock data.

Example: :Config:AIM 2000, 1, 1, 2500, 0.000000, 0.000000

QUERY: **:Configure:AIM?**

→<Update_Freq>,<master frame axis>,<master 10k axis>,<sw cap filter freq>,<clockAdjust>,
<aim delay>

EXAMPLE: :Config:AIM?
→ 2000, 1, 1, 2500, 0.000000, 0.000000

4.2.2 CONFIGURE:ANALOG[?]

The Configure ANALog Output command defines the use of the programmable analog outputs of the Axis Interface Module (AIM). The analog output number <channel> is specified as an ASCII integer (1 to 6); channels 1 and 2 are used for general purpose outputs which appear on an analog I/O panel using BNC connectors. Channels 3 and 4 may be used as extra user outputs or as servo outputs for commutation of a second 3-phase motor. Channels 5 and 6 are dedicated servo actuator outputs. Channels 7 and 8 are strictly dedicated to setting the transducer drive reference and are not accessible via this command.

COMMAND: **:Configure:Analog <axis>,<channel>,<variable>,<max>,<min>,<auto zero>**

- **<axis>** [NR1] specifies which axis (hardware AIM D/A output) is being configured.
- **<channel>** [NR1] {1..6} identifies which D/A channel is being configured.
- **<variable>** [NR1] System Variable number; where the axis "sub-system" supplying the variable is MOD_{1000}(<variable>), and the specific variable is REM_{1000}(<variable>). ** If variable #'s < 1000, then <axis> will be used; this simplifies commands, and is otherwise required for Acutrol Act2000 compatibility. Variable specifications proceeded by "S-" are interpreted as Supervisor variables, allowing Supervisor variables to be output in axis D/A channels. Also, variables of other axes can be specified; this feature greatly adds to the flexibility of analog outputs.
- **<max>** [NR2] the value of <variable> scaled in the user defined engineering units that will result in an analog output of +10 volts. Values of <variable> greater than <max> will result in an output of +10 volts.
- **<min>** [NR2] the value of <variable> scaled in the user defined engineering units that will result in an analog output of -10 volts. Values of <variable> less than <min> will result in an output of -10 volts.
- **<auto zero>** [Enum] {Enable | Disable} enables the auto zeroing of the analog output whenever the corresponding axis is not in a servo mode. This feature only applies to the four "servo" outputs {3, 4, 5, and 6}.

EXAMPLE: :CON:ANALOG 1,5,1164,10.000000,-10.000000,Enabl

The compensated SERVO_ERRor variable, x164 from axis 1 is output on Ch. 5; this output will be zero whenever axis 1 servo is active.

:Configure:Analog 2,1,1083,110.000000,90.000000,Disable

Axis 1 variable x083 (Estimated Rate) is output on Axis 2 (channel 1) Front Panel BNC - **Analog Out1**. The output is always enabled and the ±10.0Volt output is scaled to produce a window of 100 ±10 degrees/second.

The query command reports the current configuration of the selected analog output.

QUERY: **:Configure:Analog? <axis>,<channel>**

→ <variable>,<max>,<min>,<auto zero>

EXAMPLE: :CON:ANALOG? 1,3

→ 1118, 10.000000, -10.000000, Disable

Analog Channel 3 on Axis1 is not in use; Variable x118 ZERO corresponds to zero volts.

Errors:

1. Invalid <axis>. Set the PARAMETER OUT OF RANGE bit in the Status Byte.
2. Invalid <channel>. Set the PARAMETER OUT OF RANGE bit in the Status Byte
3. Invalid <variable>. Set the PARAMETER OUT OF RANGE bit in the Status Byte
4. <max> Greater than Variable Full-scale range? No error, this is a valid condition. The result is an output range less than +10 volts.
5. <min> Less than Variable Full-scale range? No error, this is a valid condition. The result is an output range greater than -10 volts.
6. Invalid <Auto-zero> Option. Set the PARAMETER OUT OF RANGE bit in the Status Byte

4.2.3 CONFIGURE:AXIS[?]

The Configure:Axis command is used to define the user names for each axis.

COMMAND: **:Configure:Axis <axis>,<axis_name>**

- <**axis**> [NR1] = {1|2|…|n}; where [n] is the maximum number of axes.
- <**axis_name**> ["(string – 10 char maximum)"] User defined name for the axis; this string must be quoted. This command can be queried by an operator interface and integrated into the legends, titles and variable names of interface panels and user reports.

EXAMPLE: :Co:Ax 1,"Azimuth"

QUERY: **:Configure:Axis? <axis>**

→ <axis_name>

EXAMPLE: :C:Ax? 2

→ "Azimuth"

4.2.4 CONFIGURE:CONDITIONER[?]

This block is used to condition raw position measurements by providing configuration parameters for offset and sense. Additionally, a hard coded gain and modulo are applied. Position conditioning is required to make raw signals compatible for correlation, and to constrain the range of position variables. The output of the coarse conditioner is scaled in the native measurements units, typically in revolutions, the full scale range of the coarse transducer. The Factory and User output variables are scaled in the engineering units that represent the demand scaling for position variables. The "hard coded" parameters associated with Gain and Modulo are defined in section Configure:Scaling.

Instance	Variable Name	Variable #	Gain	Modulo
Coarse	V_Coarse_Cond	x207	1	1 / 0
Factory	V_Fine_Cond	x107	Pos_SF	FS_Pos / Pos_Offset
User	V_Posn_Fbk	x166	1	FS_Pos / Pos_Offset

COMMAND: **:Configure:Conditioner <axis>,<Conditioner>,<pos_off>,<sense>,<pos_var>**

- **<Conditioner>** [enum] {coarse | factory | user} Specifies one of three instances of the Conditioner module.
- **<pos_off>** [NRf] is added to the raw encoded position and modulo adjusted so that any arbitrarily position can be set as zero. <pos_off> is scaled with the same units as the output position variable
- **<sense>** [char] {+|-} this argument specifies if the encoded position is inverted prior to being output.
- **<pos_var>** [NR1] Specifies the System variable that is to be conditioned.

EXAMPLE: :Con:Cond 1,coarse,0.000200,+,1085

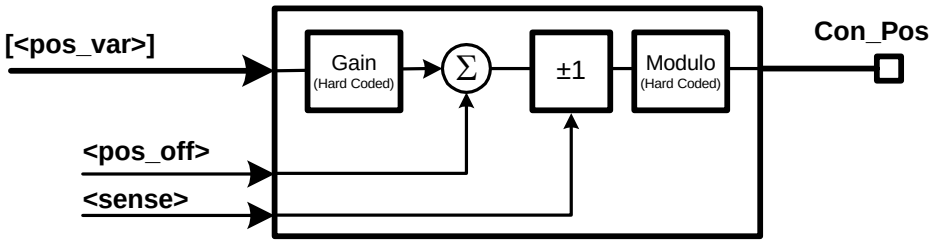
:C:Con 1,Factory , 8.804E+01, -, 1084

QUERY: **:Configure:Conditioner? <axis>,<coarse|factory|user>**

EXAMPLE: :Conf:Cond? 1,coarse

➔ 2.000000000000000E-04, +, 1085

Position Conditioner



4.2.5 CONFIG:CORRELATION[?]

Correlation is the process of combining coarse and fine resolution position information into a composite position that has the precision of the (fine) mesurement data, and the range of the (coarse) reference data. The correlation algorithm has a tolerance to correlation error (encoding and offset combined) corresponding to ±½ the range of the fine measurement position (± 0.5° in the case of a 360 speed Inductosyn). If the correlation error (either statically or dynamically) exceeds the correlation range, a miss-correlated (discontinuous) position will be output. It is not always possible to detect that a miss-correlation has occurred, but when it is detected, it will be reported in the ACP status. When correlation proceeds properly, the composite position is continuous over the range of the (coarse) reference data and consistent with the (fine) measurement data.

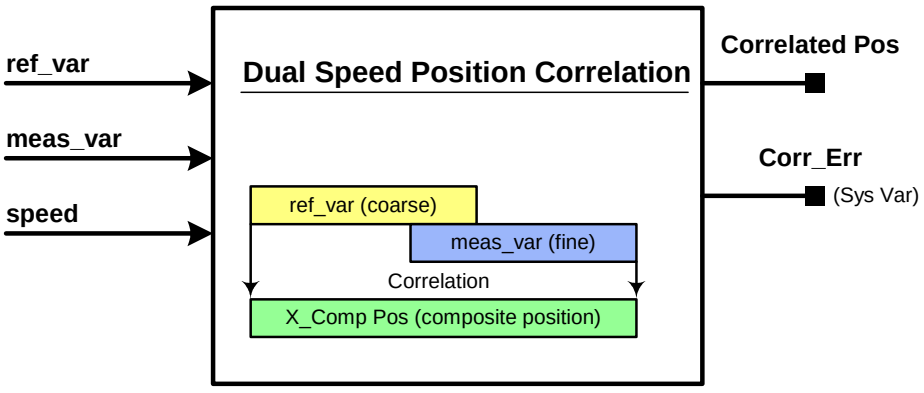
The correlation block is configured by specifying a (coarse) reference position <ref_var>, a ”raw” transducer measurement <meas_var>, and a relative scale factor between the reference and the feedback <speed>. The overlap range of the correlation is centered by adjusting the coarse transducer position offsets; coarse <pos_off> (see section 4.2.20) in the case of ratiometric transducers.

The correlated position is output as a composite of the reference and the measurement positions. This result is made available to other processes by mapping the output for each of the Dual Speed Correlation modules to System variables.

A second System variable is output as the correlation error for test/calibration purposes. This signal is computed as the difference between the correlated position and the reference position <ref_var>.

System Variable	Mnemonic	Variable #
First Correlated Position	V_Coarse_Fbk	x163
Second Correlated Position	V_Fine Fbk	x084
First Correlation Error	V_Coarse_Err	x086
Second Correlation Error	V_Fine_Err	x109

Note, the correlated position and the reference position are always scaled with the same weight; the <speed> parameter defines the relative sensitivity between the reference and measurement positions.



COMMAND: **:Configure:Correlation <axis>,<First|Second>,<speed>,<ref_var>,<meas_var>**

- **<axis>** [NR1] = {1|2|…|n}; where [n] is the maximum number of axes.
- **<First|Second>** [enum] {First, Second} specifies which correlation module to address.
- **<speed>** [NRf] Relative sensitivity between the reference position and the measurement variable. For rotational systems, speed generally needs to be an integer. However, for linear systems, <speed> could be a real number which matches sensors scaled with different measurement units (e.g. inches/centimeters or mm linear Inductosyn and a analog scaled string pot).
- **<ref_var>** [NR1] Specifies any System variable.
- **<meas_var>** [NR1] Specifies any System variable.

EXAMPLE: :C:Cor 1,Sec, 360, 1163, 1108

This example demonstrates the configuration of fine (second) correlation on axis 1. The reference variable is the Composite Estimated Position (V1163), and the measurement variable is the Raw Fine Position Feedback (V1108). The fine sensor is scaled by 1/360 of full scale. If full scale is 1 revolution as in most rotary systems, then this fine sensor has a range of ±0.5°.

QUERY: **:Configure:Correlation? <axis>,<first|second>**

→ <speed>,<ref_var#>,<meas_var#>

EXAMPLE: :Config:Corr? 1,First

→ 1.000000000000000E+00, 1168, 1207

:C:Cor? 1,Sec

→ 3.600000000000000E+02, 1163, 1108

4.2.6 CONFIGURE:DEMANDSUM[?]

This command is used to specify a System variable that will be offset, scaled, and summed with one of the demand states (position, rate, acceleration, or jerk). This provides a means of including motion demands from other sources such as analog inputs, measured states of other axes, and special axis function blocks. Once the <var>, <gain>, and <offset> are configured, these arguments can be dropped when enabling/disabling a source.

System Variable	Mnemonic	Variable #
Position Demand Sum	Var_Posn_Dmd	x264
Rate Demand Sum	Var_Rate_Dmd	x265
Acceleration Demand Sum	Var_Accel_Dmd	x266
Jerk Demand Sum	Var_Jerk_Dmd	x267

COMMAND: **:Configure:DemandSum <axis>,<state>,<enable>,<var>,<gain>,<offset>**

- **<axis>** [NR1] = {1|2|…|n}; where [n] is the maximum number of axes.
- **<state>** [enum] {Position, Rate, Acceleration, Jerk} defines the motion state that is being modified
- **<enable>** [enum] {Enable | Disable} Enables or disables this function; can be issued without altering the configuration.
- **<var>** [NR1] Specifies any System variable (even from another axis).
- **<gain>** [NR2] Specifies the gain that scales a variable to the demand scaling of the selected motion <state>.
- **<offset>** [NR2] defines an constant offset (in demand scaled units) that is additionally summed with the selected motion <state>.

EXAMPLE: :Configure:DemandSum 2,P,E,2089,18,0

The axis 2 "Analog Feedback 1" (variable 2089) is summed with the Position Demand on the same axis after being scaled to have a range of ±180° (18*±10 volts; assuming that position is scaled in degrees). No offset is applied.

:Config:Dem 2,P,Disable (Disable the analog input configured previously.)

:Config:Dem 1,R,Enable,2090,100,0.022

In this example, axis 1 Rate Demand is summed with "Analog Feefback 2" from axis 2. The analog input has a sensitivity of 100°/s/volt and an offset of 0.022°/s is applied to cancel a bias in the user supplied analog signal. The demand summer is configured and enabled.

QUERY: **: Configure:DemandSum? <axis>,<state>**

→ <enable>,<var>,<gain>,<offset>

EXAMPLE: :Configure:DemandSum? 2,P

→ D,2090,100,0.022

4.2.7 CONFIGURE:DIGITALIO[?]

This command is used to configure the (two) BNC connectors (Digital I/O 1, 2) associated with the user I/O located on the front panel of the Operator Interface. General Purpose Input/Output pins (GPIO 6:7) from the Axis Interface hardware provide the digital ports to input or output selected signals. Each pin is independently configured for data direction. The specific signal associated with each pin, when used as an output, is also configurable. The states of these pins are reported in Local_Status4 regardless of whether a pin is programmed for input or output.

GPIO 6:7 have unrestricted access so that an operator has control of the selection of the digital signals on the front panel BNCs. The other pins GPIO 0:5 cannot be changed by this command. Section 4.2.12 Configure:GPIO[?] provides a factory-restricted command to configure all GPIO pins.

This table summarizes the configuration options for the GPIO pins.

GPIO pin	I/O Name	Direction Control	Default Direction	Signal Control
7	Digital I/O 1	User	Output	User
6	Digital I/O 2	User	Output	User
5	Aim Relay 3	XML	Output	Factory
4	Aim Relay 2	XML	Output	Factory
3	AIM Relay 1	XML	Output	Factory
2	AIM INTLK 3	XML	Input	Factory
1	AIM INTLK 2	XML	Input	Factory
0	AIM INTLK 1	XML	Input	Factory

COMMAND: :Configure:DigitalIO <axis>,<DigIO>,<signal>,<direction>

- **<axis>** [NR1] specifies which axis is being configured.
- **<DigIO>** [NR1] {0|1} specifies which Digital I/O BNCis being configured (BNC 1 or 2 respectively).
- **<signal>** [NR1] {0..15} Defines the discrete signal to be output; refer to the GPIO Selection Table below.
- **<direction>** [enum]{Input | Output} Defines the data direction (I)nput to Acutrol or (O)utput from Acutrol.

EXAMPLE: :Config:DigitalIO 2,7,5,Output (Configure Digital I/O 2 on axis 2, to output signal #5)

EXAMPLE: :Config:DigitalIO 3,6,,Input (Configure Digital I/O 1 on axis 3, as an input.)

Note: the omitted <signal> (,,) parameter signifies that the multiplexed selection is not changed.

QUERY: **:Configure:DigitalIO? <axis>,<GPIO>**

→ <signal>,<direction>

EXAMPLE: :Config:Dig? 1,7

→ 5,Output Digital I/O 2 is configured to output signal#7on axis 1.

:C:dig? 3,0

→ 0, Output Digital I/O 1 is configured to output signal#0 on axis 3.

4.2.8 CONFIGURE :DISCRETE[?]

This command is devised to provide the user/programmer a means to control the state of discrete outputs of the Acutrol3000 controller. Since all discrete I/O is controlled through the Axis Interface Modules (AIM), this command has both axis and system control implications. Each AIM in a system has a programmable register [PROG_OUT] whose value can be set by this command. This command has the added feature that the entire PROG_OUT register can be changed with one command, or just one bit can be changed without affecting other bits in the register.

The AIM module has 8 general-purpose I/O pins [GPIO] that can be configured as either inputs or outputs. If a GPIO pin is configured as an output, then it can also be configured to reflect the discrete state of 1 of 16 (FPGA) internal signals associated with that specific pin. Specific bits of the PROG_OUT register can be selected to define the state of a general-purpose discrete I/O pin. By changing the state of this bit in PROG_OUT, discrete outputs can be controlled. Configuration of the GPIO pins is described in CONFig:GPIO Sect (4.2.12).

This command is compatible with ACT2000 with the proviso that the MIS1 argument corresponds to axis 1; therefore, only discrete controls from axis 1 can be used.

The state of this command is not preserved in the XML configuration file. However the configuration of the discrete I/O is preserved by the saved states of the PROG_OUT register, and the configuration of the GPIO pins.

When using this command to control discrete outputs, the programmer must ensure that the I/O pin attempting to be controlled is properly configured; furthermore, the configuration options for using specific GPIO pins is constrained by the hardware options of the Transition panel and any other system requirements for discrete (interlock) inputs or (relay) outputs. A detailed explanation of how the GPIO pins are configured and programmatically used to control discrete outputs is found in **TM-9385 Safety Interlocks, STATUS, and Discrete Control Integration Procedure.**

COMMAND: **:Configure:Discrete [<axis>|ALL],<closure>[,<state>]**

- **<axis>** [NR1]
  - ALL - address the PROG_OUT registers of all axes present in the system
  - 1 to n = Axis specific reference to a single axis PROG_OUT register
  - 0 = undefined; no response produced.
- **<closure>** [NR1]
  - If <state> is included, then <closure>={0…15} specifies the bit of PROG_OUT register which is changed to the value of <state>.
  - If <state> is not included, then <closure> is a 16 bit binary word that sets all 16 bits of the PROG_OUT register.
- **<state>** [NR1] = {0 | 1} defines the logical state of the PROG_OUT(<closure>) bit; this argument is optional.

EXAMPLE: :Config:Discrete 1,3,1 (set PROGOUT bit 3 on axis 1)

:Config:Discrete All,5,0 (clear PROGOUT bit5 on all axes)

:Con:Disc 2,257 (Set bits 0 and 8 and clear all others on axis 2 only; note,257 decimal = 000 0001 0000 0001 binary)

The DISCrete query command will return the state of the selected bit of the PROG_OUT register defined by the <closure> argument. If <closure> is omitted, then <closure> is returned and represents the state of the 16-bit PROG_OUT register.

QUERY: :Configure:Discrete? <axis>[,<closure>]

→ <state>

EXAMPLE: :CONF:DISC? 1,3

→ 1 (bit 3 on axis 1 is TRUE, closed)

:C:Dis? 2

→ 257 (only bits 0 and 8 are TRUE on axis 2, all other bits are FALSE)

Errors:

1. Invalid <axis>. Set the PARAMETER OUT OF RANGE bit in the Status Byte.
2. Invalid <closure> , <closure> being a 16 bit value when <bit> is sent. Set the SYNTAX ERROR bit in the Status Byte.
3. Invalid <bit>. Set the PARAMETER OUT OF RANGE bit in the Status Byte.

4.2.9 CONFIGURE:EVENT[?]

This command is used to configure Event pulses, which are generated as an axis moves through evenly spaced positions (Events). The event positions are indirectly programmed by defining both, the number of events over the full-scale range of the absolutely encoded position and the initial offset position of the first pulse. Note, the majority of systems have a full-scale range of 1 revolution so that setting 360 pulses full scale will result in 1° position intervals. If the system has been scaled for multiple revolutions, the number of pulses would have to be adjusted accordingly (e.g. 10 revolutions full scale requires 3600 pulse events to produce 1° position intervals). The full scale range is defined using the Configure:Scaling[?] command in section 4.2.26.

The maximum pulse rate is equal to the reciprocal of the Frame update period, e.g. an update period of 1 ms allows a maximum of 1000 pulses per second, and an update period of 0.2 ms permits the generation of up to 5000 pulses per second. The pulse width is also programmable and should never be wider than one-half of the Frame period. The Event pulse can be set to strobe high or low and either the leading or trailing edge of the pulse can be used to measure time between pulses without compromising rate accuracy measurements. The leading edge of the pulse corresponds to the time that the position event occurred and should be used to trigger external processes that would be compromised by a position event phase error.

COMMAND: **:Configure:Event <axis>,<event>,<pulses_fs>,<pulse_width>,<sense>,<pulse_pos_offset>**

- **<axis>** [NR1] specifies which axis is being configured.
- **<event>** [NR1] {1, 2} specifies which of two hardware pulse generation channels is addressed.
- **<pulses_fs>** [NR1] Defines the integer number of pulses that are evenly distributed over the absolute full scale position range for an axis.
- **<pulse_width>** [NR2] {0 to 1024 uS} This sub-command sets the pulse width of an event pulse. The <pulse_width> is independent for each axis, but common for both events on the same axis. The pulse width resolution is 31.25 nano-sec.
- **<sense> [Char]** {+} => High pulsing low and { - } => Low pulsing high
- **<pulse_pos_offset>** [NR2] This value specifies the position offset between the user defined zero position and the "base" position pulse. The base pulse refers to the position pulse that is generated at zero position with the <pulse_pos_offset> = 0. This is easily visualized when only one pulse is produced over the full-scale range. . Note that offset is effectively modulo'd by pulses_fs / position Full Scale.

EXAMPLE: :Config:Event 1,1,1,20,+,45.125

Sets Event pulse 1 on axis 1 to produce 1 pulse per revolution in the case of a continuous rotation system. The pulse has duration of 20 microseconds, and is normally high pulsing low. The pulse will be generated with an offset of 45.125 degrees, assuming position demand scaling is in degrees.

:con:ev 1,2,360

Sets Event pulse 2 on axis 1 to produce 360 pulses per revolution; this corresponds to 1° pulses in the case of a continuous rotation system. Pulse width, mode, and offset are not changed from the last **:Config:Event** command or the initialized state.

COMMAND: **:Configure:Event? <axis>,<event>**

→ <pulses_fs>,<pulse_width>,<sense>,<pulse_pos_offset>

EXAMPLE: :configure:Event? 1,1

→ 1,20,+,45.125

EXAMPLE: :C:E? 1,2

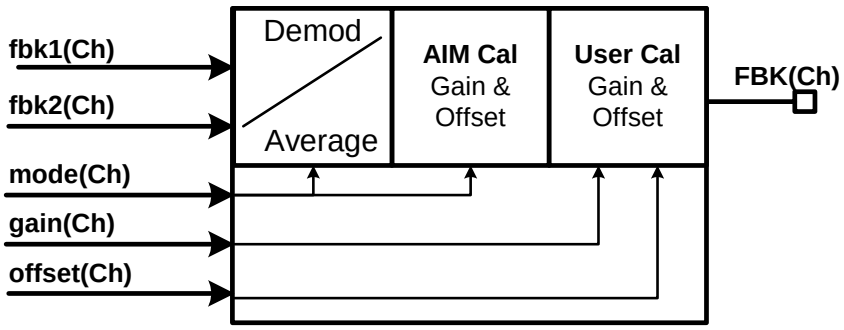
→ 360,20,+,0

4.2.10 CONFIGURE:FBKCHANNEL[?]

This command is used to enable and scale the four transducer data acquisition channels. For ratiometric position encoding, two channels are used in conjunction with the ratiometric encoding module to produce a continuous measurement. A single channel can be used with carrier modulated displacement transducers such as Linear Voltage Differential Transducer (LVDT). This module can be configured to demodulate an AC carrier modulated signal or average a DC feedback signal. The FBKChannel provides scale and offset calibration for the associated System variable that is generated.

The System variables are assigned as follows:

System Variable	Mnemonic	Variable #
Feedback Channel 1	V_COARSE_SIN	x203
Feedback Channel 2	V_COARSE_COS	x204
Feedback Channel 3	V_FINE_SIN	x103
Feedback Channel 4	V_FINE_COS	x104



COMMAND: **:Configure: Fbkchannel <axis>, <channel>,<mode>,<gain>,<offset>**

- **<axis>** [NR1]
- **<channel>** [NR1] {1|2|3|4} Select the AC feedback channel to be configured.
- **<mode>** [enum] {AC | DC | None} enables the configuration of this channel.
  "Ac" => AC feedback; signal is demodulated
  "Dc" => DC feedback; signal is averaged
  "None" => Not enabled; corresponding System variable is set to zero
- **<gain>** [NRf]] converts a peak-to-peak sinusoidal or DC signal in volts to the corresponding System variable scaled in engineering units.
- **<offset>** [NRf] provides a means to bias or offset the output variable, and is scaled in the same units as the FBKCh variables.

EXAMPLE: :Con:Fbk 1,1, AC, 1.000000, 0.006

COMMAND: :Configure:FbkChannel? <axis>,<channel>

→<mode>,<gain>,<offset>

EXAMPLE: :Conf:Fbk? 1,1

→ Ac, 1.000000000000000E+00, 6.000000000000000E-03

4.2.11 CONFIGURE:FILTER[?]

The Filterer command provides a means of altering the parameters of the second order digital filter block used in the observer and controller servo structures. The digital filter implements the discrete time transfer function F(z) as the ratio of two binomial equations, where "Z" is the complex discrete time frequency. The precise formulation is defined in the Acutrol3000 servo documentation (reference required).

$$F(z) = \left[\text{gain} \cdot \frac{z^2 + n_1 \cdot z + n_0}{z^2 + d_1 \cdot z + d_0} \right]_{\text{limit}}$$

In the current implementation of filters and summers, the Summer(n) ouptut is routed to the corresponding Filter(n) input. Summers (and filters) are sequentially executed in two blocks. The first (in time) is the Observer block consisting of Summers/Filters 20 through 24. The second block is executed at a later time after all controller inputs are collected, and consists of Summer/Filters 1 through 15. Summer/filter blocks are computed in sequence from lower to higher number.

A <zero> and <alternate> arguments define the behavior of the filter as the axis controller assumes various servo operational states. An Alternate set of filter coefficients may be selected under program control to change the dynamic behavior of the Observer and the Servo compensations. A feature is provided to zero a filter's output whenever the axis servo is not servoed.

COMMAND: **:Config:Filter <axis>,<filter>,<n0>,<n1>,<d0>,<d1>,<gain>,<limit>[,<zero>[,<alternate>]]**

- **<axis>** [NR1] = { 1 | 2 |….| n }; where [n] is the maximum number of axes.
- **<filter>** [NR1] = { 1 | 2|….| 15 }; controller filter numbers.
- **<filter>** [NR1] = { 20 | 21 |…| 24 }; observer filter numbers.
- **<n0>, <n1>, <d0>, <d1>** [NRf]; polynomial digital filter coefficients described in the filter equation above.
- **<gain>** [[NRf]; the overall constant that normalizes the gain of the transfer function.
- **<limit>** [NRf]; sets the maximum positive and negative values that can be produced at the output of the filter.
- **<zero>** [enum] {Enable | Disable} Enable automatically zeros filter internal states whenever axis is not servoed.
- **<alternate>** [enum] {Enable | Disable} Enables using alternate filter coefficients .

EXAMPLE: :CONF:FILT 3,20,1.0,0.939101,-1.618034,-1.568887,0.969234,10.0,D,E

A query of the FILTer command allows the current filter coefficients to be retrieved.

QUERY: **:Config:Filter? <axis>,<filter>**

→ <n0>,<n1>,<d0>,<d1>,<gain>,<limit>,<zero>,<alternate>

EXAMPLE: :C:Fi? 3,20

→ 1.000000,0.939101,-1.618034,-1.568887,0.969234,10.000000,Disable,Enable

Errors:

1. Invalid <axis>. Parameter out of Range Error
2. Invalid <filter> Parameter out of range error.

The Output of each Filter produces a corresponding System Variable and is summarized in the table below:

System Variable	Mnemonic	Variable #
Control Filter1	V_CTRL1_OUT	X031
Control Filter 2	V_CTRL2_OUT	X033
Control Filter 3	V_CTRL3_OUT	X035
Control Filter 4	V_CTRL4_OUT	X037
Control Filter 5	V_CTRL5_OUT	X039
Control Filter 6	V_CTRL6_OUT	X041
Control Filter 7	V_CTRL7_OUT	X043
Control Filter 8	V_CTRL8_OUT	X045
Control Filter 9	V_CTRL9_OUT	X047
Control Filter 10	V_CTRL10_OUT	X049
Control Filter 11	V_CTRL11_OUT	X051
Control Filter 12	V_CTRL12_OUT	X053
Control Filter13	V_CTRL13_OUT	X055
Control Filter 14	V_CTRL14_OUT	X057
Control Filter 15	V_CTRL15_OUT	X059
Observer Filter 20	V_OBS1_OUT	X061
Observer Filter 21	V_OBS2_OUT	X063
Observer Filter 22	V_OBS3_OUT	X065
Observer Filter 23	V_OBS4_OUT	X067
Observer Filter 24	V_OBS5_OUT	X069

4.2.11.1 CONFIGURE:FILTER:ALTERNATE[?]

This command is used to define coefficients and behaviors for an alternate filter configuration. A filter will use these values instead of the default valued if enabled as defined by the **Config:Filter <alternate>** parameter, see section 4.2.11.

COMMAND: **:Config:Filter:Alternate <axis>,<filter>,<n0>,<n1>,<d0>,<d1>,<gain>,<limit>[,<zero>]**

- **<axis>** [NR1] = { 1 | 2 |….| n }; where [n] is the maximum number of axes.
- **<filter>** [NR1] = { 1 | 2|….| 15 }; controller filter numbers. **<filter>** [NR1] = { 20 | 21 |…| 24 }; observer filter numbers.
- **<n0>, <n1>, <d0>, <d1>** [NRf]; polynomial digital filter coefficients described in the filter equation above.
- **<gain>** [[NRf]; the overall constant that normalizes the gain of the transfer function.
- **<limit>** [NRf]; sets the maximum positive and negative values that can be produced at the output of the filter.
- **<zero>** [enum] {Enable | Disable} Enable automatically zeros filter internal states whenever axis is not servoed.

EXAMPLE: :CONF:FILT:Alt 3,20,1.0,0.939101,-1.618034,-1.568887,0.969234,10.0,Enable

A query of the Alternate FILTer command allows the current alternate filter coefficients to be retrieved.

QUERY: **:Config:Filter:Alternate? <axis>,<filter>**

→ <n0>,<n1>,<d0>,<d1>,<gain>,<limit>,<zero>

EXAMPLE: :C:Fi:A? 3,20

→ 1.000000,0.939101,-1.618034,-1.568887,0.969234,10.000000,E

Errors:

1. Invalid <axis>. Parameter out of Range Error
2. Invalid <filter> Parameter out of range error.

4.2.11.2 CONFIGURE:FILTER:SELECT[?]

This command is used to activate the alternate filter coefficients defined in Configure:Filter:Alternate section 4.2.11.1. The alternate compensation must be enabled on a filter by filter basis using the <alternate> argument in the Configure:Filter command, section 4.2.11. The default state can be set to either Normal or Alternate; however, both sets of filter coefficients have the same compensation features.

An additional argument <And|Or|Ignore> permits the incorporation of an external discrete signal <IO> into the logic, to provide external control.

COMMAND: :Config:Filter:Select <axis>,<Normal|Alternate>,<And|Or|Xor|Ignore>,<IO>

- **<axis> [NR1] = { 1 | 2 |….| n }; where [n] is the maximum number of axes.**
- **<Normal|Alternate>** [enum] {Normal | Alternate} Select either the Normal or Alternate gain set of filter coefficients. As stated above, only enabled filters will switch to the alternate coefficients when <Alternate> is selected.
- **<And|Or|Ignore>** [enum] {And | Or | Ignore} This argument defines the logical operation that is applied to the external Alternate Filter discrete control signal.
- **<IO>** [NR1] {0:9} Specifies which discrete I/O signal is used to enable the alternate filter coefficients. The options are summarized below:

<IO>	0	1	2	3	4	5	6	7	8	9
Source	GPIO0	GPIO1	GPIO2	GPIO3	GPIO4	GPIO5	GPIO6	GPIO7	DE0	DE1

Selecting the desired gain coefficients requires proper configuration of both the <Normal|Alternate> ACL parameter and the logical operator <And|Or|Xor|Ignore>in conjunction with the external <IO> signal. The table below defines the configuration logic where “N” refers to the Normal set of coefficients, and “A” refers to the Alternate coefficients.

Note, the logic of the discrete input can be changed by the Interlock:Logic command, see section 4.5.4.

Discrete <IO> Logic		“AND”		“OR”		“XOR”		“IGNORE”	
		1	0	1	0	1	0	1	0
ACL	Normal	N	N	A	N	A	N	N	N
	Alternate	A	N	A	A	N	A	A	A

A query of the FILTer Select command allows the current configuration to be retrieved.

COMMAND: **:Config:Filter:Select? <axis>**

→<Normal|Alternate>,<And|Or|Xor|Ignore>,<IO>

EXAMPLE: :C:Filter:S? 3

→ Normal, And, 3

4.2.12 CONFIGURE:GPIO[?]

This command is used to configure the eight discrete General Purpose Input/Output (GPIO) pins on each axis the AIM boards. Each pin is independently configured for data direction. The specific signal associated with each pin when used as an output is also configured. The states of these pins are reported in **Local_Status4** regardless of whether a pin is programmed as an input or an output.

The GPIO pins are configured on initialization, setting the direction (input/output) and the multiplexed signal for each pin. GPIO 3, & 5 are generally mapped as discrete control outputs, and GPIO 0,1, 2 & 4 are mapped as interlock inputs; this is in compliance with the default hardware configuration of the Acutrol Transition panel. GPIO 6 & 7 pins are mapped to the front panel Digital IO BNC connectors.

In the GUI, GPIO 6:7 have unrestricted access so that an operator has control of the selection of the digital signals on the front panel BNCs. The other pins GPIO 0:5 may be changed as well if the operator has configuration privileges. If one of these pins is an output, then it cannot be re-configured by the general User. When direct ACL commands are used, there are no restrictions on the configuration of GPIO pins.

GPIO pins that are used for interlocks (inputs), function independently of the associated output multiplexers, leaving the selected signals available for indirect routing to the front panel User I/O BNCs. This increases the number of signals that can be selected (in the default case) from 32 to 74. Table 5.2.12 defines the signals that can be selected by the GPIO output multiplexers.

The following table summarizes the default configuration for the GPIO pins.

GPIO pin	I/O Name	Direction Control	Default Direction	Signal Control
7	Digital I/O 1	User	Output	User BNC IO-2
6	Digital I/O 2	User	Output	User BNC IO-1
5	Aim Relay 3	XML	Output	Factory
4	Aim Relay 2	XML	Output	Factory
3	AIM Relay 1	XML	Output	Factory
2	AIM INTLK 3	XML	Input	Factory
1	AIM INTLK 2	XML	Input	Factory
0	AIM INTLK 1	XML	Input	Factory

COMMAND: **:Configure:GPIO <axis>,<GPIO>,<signal>, <direction>**

- **<axis>** [NR1] specifies which axis is being configured.
- **<GPIO>** [NR1] {0:7} specifies which GPIO pin is being configured.
- **<signal>** [NR1] {0..15} Defines the discrete signal to be output; refer to the GPIO Selection Table below. This argument must be specified, even if the direction is Input.
- **<direction>** [enum] {Input | Output} Defines the data direction Input to Acutrol or Output from Acutrol.

EXAMPLE: :Config:GPIO 2,7,5,Output (Configure GPIO7 on axis 2, to output the FS_LOCAL signal.)

EXAMPLE: :Config:GPIO 2,3,2,Output (Configure GPIO3 on axis 2, to output signal #2 (PGOUT3) to control AIM Relay3.)

EXAMPLE: :Config:GPIO 1,0,Input (Configure GPIO0 on axis 1, as an input.)

QUERY: :Configure:GPIO? <axis>,<GPIO>

→ <signal>, <direction>

EXAMPLE: :Config:gpio? 1,6

→ 7,Output

Pin GPIO6 is configured to output signal#7on axis 1.

:Con:GPIO? 1,2

→ 3,Input

Pin GPIO2 on axis 1 is configured as an input.

Table 5.2.12 GPIO Selection

GPIO Selection Table				
* indicates default data direction				
Signal	**GPIO0 - Input***	**GPIO1 - Input***	**GPIO2 - Input***	**GPIO3 - Output***
0	LO	LO	LO	LO
1	E_CLOCK	E_DIRECTION	SDL_FAIL	DIOO
2	E_REF	E_AQUAD	E_BQUAD	PGOUT0
3	GP3_STROBE	GP4_STROBE	CLK1M	PGOUT3
4	DEC0	RREQ	ADDBLK	PGOUT6
5	DEC1	L_BACK	SA1	PGOUT9
6	AVR_INT0	BLCK	CLK_WDOG/	PGOUT12
7	AVR_INT1	PC_OUT_ENBL	FIFO_READ	PGOUT15
8	DSD	DIP	RIP	GRIP
9	DSE	FIFO_RESET	RDR	DE0
10	GCIP	IOR	IOW	MASTER_INTLK
11	AEN	I/OCS16/	WREQ	RT_SYNC
12	CONV_ST/	DMA_INT	IRQ	EVP1
13	SELA2	SELB2	DMA_DONE	SIL_FAULT
14	WRITE_MUXPORT	FIFO_FULL	SHBE	CL_INTLK_STROBE
15	DSF	CSA/	SYNC_TIME_CS	GPD_OUT0
Signal	**GPIO4 - Output***	**GPIO5 - Output***	**GPIO6 - User Output***	**GPIO7 - User Output***
0	LO	LO	LO	LO
1	DIOC	DIOL	PGOUT4	PGOUT5
2	PGOUT1	PGOUT2	PGOUT6	PGOUT7
3	PGOUT4	PGOUT5	DE0	DE1
4	PGOUT7	PGOUT8	GP1_STROBE	GP2_STROBE
5	PGOUT10	PGOUT11	FS_IN	FS_LOCAL
6	PGOUT13	PGOUT14	10KREF_IN	CLK10K
7	GP1_STROBE	GP2_STROBE	HW_FAULT	CLK10HZ
8	GP3_STROBE	GP4_STROBE	DMA_SYNC_ERROR	10K_LOCK_FAULT
9	DE1	MICRO_INTERRUPT	ALIVE	CLK100K
10	INTLK1	INTLK2	DL_TRIG	T/C
11	PA_ACK0	PA_FAULT0	DACK	DRQ
12	EVP2	SP_RESET	HPD	HSP
13	HW_FAULT	FS_LOCAL	GPD_OUT1	GPD_OUT0
14	OP_INTLK_STROBE	FAULT_RESET_STROBE	GPD_OUT3	GPD_OUT2
15	GPD_OUT1	GPD_OUT2	GPD_OUT5	GPD_OUT4

4.2.13 CONFIGURE:LUT[?]

Look-up tables (LUT) are used in the Acutrol3000 to correct for deterministic errors that are identified in the measurement and actuation systems. The LUTs are initialized by an initialization file and can be modified in situ by adding harmonic terms to the table data series.

This command defines the input System variable <in_var> that is to be corrected and the reference System variable <ref_var>, which is the basis for interpolation of an error term. The error term is added to the input variable and the result is output as a corrected System variable. The error term is also output as a System variable, LUTerror().

Look-up-tables are automatically loaded at boot time, but are only saved on demand.

COMMAND: :Configure:LUT <axis>,<table>,<in_var>,<ref_var>,<ref_range>

- **<table>** - Coarse | Position | Cogging | 3Phase | User.
- **<in_var>** [NR1] Reference to the System variable that is modified by the look-up-table.
- **<ref_var>** [NR1] Reference to the System variable that is used to index through the look-up-table.
- **<ref_range>** [NR2] defines a range that corresponds to the limits of the look-up-table. This argument is scaled the same as the variable specified by <ref_var>.

EXAMPLE: :Config:LUT 1,Position,1107,1107,360.000000

EXAMPLE: :Config:LUT 2,Coarse,1085,1085,360.000000

QUERY: :Configure:LUT? <axis>,<table>

→<in_var>,<ref_var>,<ref_range>

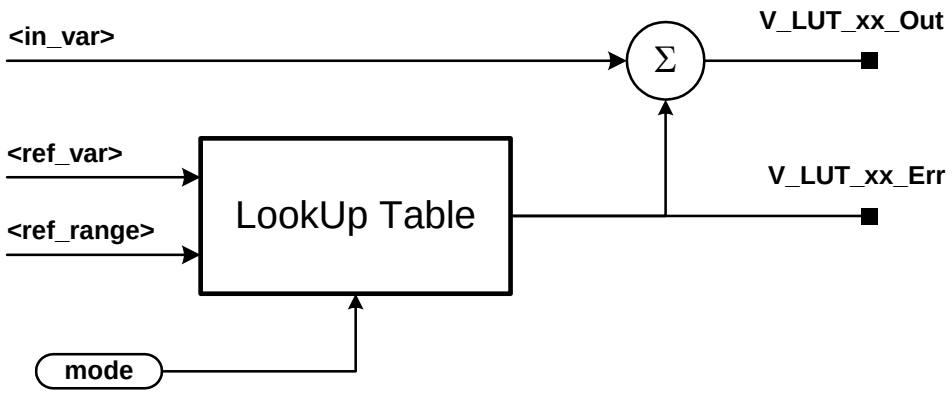
EXAMPLE: :Config:LUT? 1,Coarse

→1085, 1085, 360.000000

Each LUT generates a new variable corresponding to the corrected value of the <in_var>. A second System variable corresponding to the Error Value of the lookup table is always computed regardless of the enable status.

Note, the Look-Up tables in the GUI are identified as follows:

Coarse LUT → Coarse Position
Position → Factory Position
Cogging → Cogging Torque
3Phase → Three Phase Motor
User → User Position



Look-UP Table Block Diagram

Look-Up Table Variables

System Variable	Mnemonic	Variable #
Corrected Coarse Position	V_LUT_Coarse_Out	X240
Coarse Position Error	V_LUT_Coarse_Err	X241
Corrected Factory Position	V_LUT_Position_Out	X242
Factory Position Error	V_LUT_Position_Err	X243
Corrected Cogging Torque	V_LUT_Cogging_Out	X244
Coarse Cogging Error	V_LUT_Cogging_Err	X245
Corrected 3-Phase Motor	V_LUT_3Phase_Out	X246
3-Phase Motor Error	V_LUT_3Phase_Err	X247
Corrected User Position	V_LUT_User_Out	X248
User Position Error	V_LUT_User_Err	X249

4.2.13.1 CONFIGURE:LUT: FOURIER

This Fourier command is used to add a sinusoidal function to the raw values stored in a LUT. Harmonic number, amplitude, and phase coefficients uniquely define the sine wave. The value of the function is calculated for each point of the LUT and is summed with the current data in the table. This process can be performed as many times as is required to build a function which models and cancels systematic errors. There is no recording of the Fourier terms that are added to the table; consequently, the only record of the composite error function is the data stored in the table. The data in the LUT is also stored in a file and is used for initialization. This data can be transferred via FTP to an external computer and analysis will reveal the harmonic content of the LUT error function.

COMMAND: **:Configure:LUT:Fourier <axis>,<table>,<harmonic>,<amplitude>,<phase>**

- **<table>** - Coarse | Position | Cogging | 3Phase | User.
- **<harmonic>** [NR2] Specifies the spatial harmonic where one cycle of the fundamental is spread over the full range of the LUT which also corresponds to <ref_range>. In general the harmonic is specified as an integer, but fractional harmonic frequencies can be used with caution.
- **<amplitude>** [NR2] Specifies the peak amplitude of the sine wave and is scaled in the same units as <in_var>.
- **<phase>** [NR2] Specifies the initial phase scaled in Degrees and normalized to the harmonic frequency (90° shifts any harmonic sine wave by ¼ of a cycle).

The sine wave function Fn is defined over the range of (n = 0 to <points>-1) as follows:

Fn = <amplitude>*sin(2pi*n*<harmonic>/<points> + <phase>*2pi / 360)

This command may not be queried.

4.2.13.2 CONFIGURE:LUT:LIST?

This command is not implemented at this time.

This command is used to return a list of all look-up tables for the specified <table> instance and axis. Each record in the list contains the file version and the corresponding comment.

COMMAND: **:Configure:LUT:List? <axis>,<table>**

➔ **<version>,<comment>[,<version>,<comment>[,…]]**

- **<table>** - Coarse | Position | Cogging | 3Phase | User.
- **<version>** is a three digit numeric reference for the LUT. "001" is the default LUT loaded on power up**.**
- **<comment>** is a ASCII string description of the LUT contents and is limited to 72 characters. Comment must be in quotes "…".

4.2.13.3 CONFIGURE:LUT:MODE [?]

The LUT **Mode** command is used to control the active state of the specified lookup table.

Command: **:Configure:LUT:Mode[?] <axis>,<table>,<mode>**

- **<table>** - Coarse | Position | Cogging | 3Phase | User.
- **<mode>** {Normal|Disable} sets the operational behavior of the LUT to the following options:

  Normal – Interpolate from LUT and sum to output variable LUT_Out(). This mode is the default state.

  Disable – LUT output is not summed to the output LUT_Out(). The Error variable is still computed

EXAMPLE: :Config:LUT:Mode 1,Position,Normal

EXAMPLE: :Config:LUT:Mode 3,Coarse,Disable

QUERY: **:Configure:LUT:Mode? <axis>,<table>**

➔ **<mode>**

EXAMPLE: :Config:LUT:Mode? 1,Coarse

➔Disable

4.2.13.4 CONFIGURE:LUT: PRESET?

This Command is used to set initial conditions for a new or existing Look-Up table. When this command is issued, any data previously defined for the table (in memory) will be destroyed and all points in the table will be initialized with the <preset value>. This command will size or re-size a table as specified by the <num points> argument. It is a good practice to save an existing table to a backup file before re-initializing. No permanent changes are made to the system until the LUT is saved to the default file, see section :Config:LUT:Save.

COMMAND: **:Configure:LUT:Preset <axis>,<table>,<num points>,<preset value>**

- **<table>** - Coarse | Position | Cogging | 3Phase | User.
- **<num points>** [NR1] Defines the size of the LUT. This argument must be specified.
- **<preset value>** [NR2] Defines the value that all points are set to by this command. This argument defaults to zero if not specified.

EXAMPLE: :Config:LUT:Preset 2,cogg,4000,0

:Config:LUT:Preset 1,Coarse,2880,0.123

QUERY: **:Configure:LUT:Preset? <axis>,<table>**

➔ <num points>,<Computed Average>

- **<num points>** [NR1] Defines the size of the LUT. This argument must be specified.
- **<Computed Average>** [NR2] Defines the average value of all points in the table.

EXAMPLE: :Config:LUT:Preset? 2,Cogging

→ 4000,0

EXAMPLE: :Config:LUT:Preset? 1,Coarse

→ 2880,0.123

Note that the <Computed Average> will return the computed average value of the table. Generally this is the same value as the <preset value> since the **:Con:LUT:Fourier** command only modifies the LUT with zero-mean sinusoids. This will not be the case if fractional harmonics (such as 0.5) are used in a calibration or if an arbitrary data series is defined.

4.2.13.5 CONFIGURE:LUT: RESTORE [?]

This command is used to transfer the contents of a LUT data file to one the LUT data structures. The numeric <version> permits restoring 1 of 999 possible LUT data files. The default file restored when the system is initialized is version (001). The actual file name syntax is [<table>.<version>.<axis>]; an example is [Coarse.001.2].

COMMAND: **:Configure:LUT:Restore <axis>,<table>,< version >**

- **<table> - [**Coarse | Position | Cogging | 3Phase | User].
- **<version>** is a three digit numeric reference for the LUT. "001" is the default LUT loaded on power up**.**

QUERY: **:Configure:LUT:Restore? <axis>,<table>**

→ <version>

4.2.13.6 CONFIGURE:LUT: SAVE

This command is used to transfer the contents of one of the LUT data lists to a file. The numeric <version> allows for storing multiple tables supporting test, backup, and operation specific calibration curves. The default file loaded when the system is initialized is version (001). The actual file name syntax is [<table>.<version>.<axis>]; an example is: [Coarse.001.2]. A comment field is provided and added to the data file to document unique table features.

COMMAND: **:Configure:LUT:Save <axis>,<table>,< version >,<"comment">**

- **<table>** - Coarse | Position | Cogging | 3Phase | User.
- **<version>** is a three digit numeric reference for the LUT. "001" is the default LUT loaded on power up**.**
- **<"comment">** is a ASCII string description of the LUT contents and is limited to 72 characters. Comment must be in quotes "…".

QUERY: **(none)**

4.2.13.7 CONFIGURE:LUT: SERIES [?]

This command is not implemented at this time.

This command provides a means to upload or download a series of data points t/from a look-up table.

Command: **:Configure:LUT:Series[?] <axis>,<table>,<data>**

- **<table>** - Coarse | Position | Cogging | 3Phase | User.
- **<data>** is definite length binary data uniformly distributed over the full scale variable range. Similar for queries.

4.2.14 CONFIGURE:MODEL[?]

This command feature provides access to the configuration parameters associated with the Observer plant model. The model provides a double integration of acceleration to produce estimates of acceleration, rate, and position. A soft clamp puts a realistic bound on the rate state to facilitate recovery in the case of loop saturation. The estimated position state is adjusted by a Range Modulus based on the configuration parameter <FS_Pos> see section 4.2.26 Configure:Scaling.

System Variable	Mnemonic	Variable #
Estimated Position	V_POSN_EST	x165
Estimated Rate	V_RATE_EST	x083
Estimated Acceleration	V_ACCEL_EST	x176

COMMAND: **:Configure:Model <axis>,<excit_var#>,<accel_FF_var#>,<pos_init_var#>,…**
…<rate_init_var#>,<ORL_lim>,<ORL_gain>

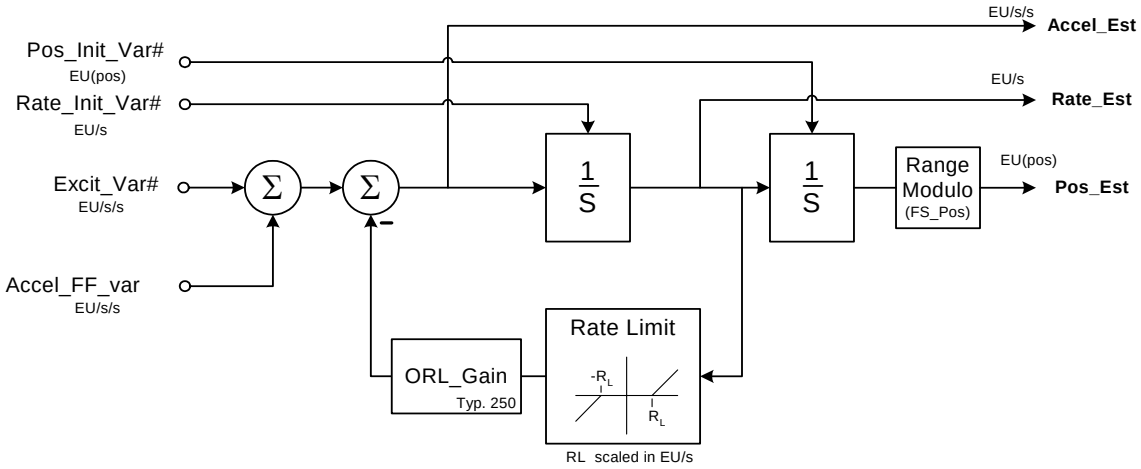
- **<axis>** [NR1] = {1|2|…|n}; where [n] is the maximum number of axes.
- **<excit_var#>** [NR1] Variable number of signal corresponding to the normalized servo error
- **<accel_FF_var#>**[NR1] Specifies the System variable used for Observer feed forward.
- **<pos_init_var#>** [NR1] Points to the variable used to initialize the position state
- **<rate_init_var#>** [NR1] Points to the variable used to initialize the rate state
- **<ORL_lim>** [NRf] Defines the rate that the soft clamp begins to have an effect on the Rate_Est .
- **<ORL_gain>** [NRf] Gain of soft clamp after <ORL_lim is reached

EXAMPLE: :Con:Model 2,2061,2043,2166,2083,400,250

Query: **:Configure:Model? <axis>**
➔ <excit_var#>,<accel_FF_var#>,<pos_init_var#>,<rate_init_var#>,<ORL_lim>,<ORL_gain>

EXAMPLE: :Con:Model? 2
➔ 2061,2043,2166,2083,400.000000,250.000000

Acutrol3000 Observer Plant Model



4.2.15 CONFIGURE:MOTIONSTATES[?]

This command is used to define which motion state System Variable is returned whenever an mnemonic ACL command is issued, e.g. Read:Position. The output of this function is a unique set of System variables, corresponding to the mnemonic references. This feature gives the user control over which internal variables are the source for monitoring the motion states, and also allows the user, using the Config:Variable command, to specify the output format independently of the source variables.

The output System variables are defined as follows:

System Variable	ACL Command Mnemonic	Factory Default Variable #
Position Output variable	Position	x082
Rate output variable	Rate	x083
Acceleration output variable	Acceleration	x176
Jerk output variable	Jerk	undefined

Note: <jerk_var> is a place keeper for a future jerk limited command processor algorithm. This argument can be omitted; when queried <jerk> will be reported as the "zero" variable x118.

COMMAND: :Configure:MotionStates <axis>,< pos_var>,<rate_var>,<accel_var>[,<jerk_var>]

EXAMPLE: :Config:Motion 2,ACL,2082,2083,2176

Query: :Configure:MotionStates? <axis>

➔< pos_var>,<rate_var>,<accel_var>,<jerk_var>

EXAMPLE: :Config:Motion 2

➔ACL,2082,2083,2176,2118

4.2.16 CONFIGURE:MOTOR[?]

This command configures the commutation software block that produces the servo error signals required to drive brushless motors. The torque command is modulated as a function of axis position and generates two system variables. The signals can be phased for two or three phase motor commutation and are output using the analog channels in the ServoA and ServoB connectors. Two identical modules permit independent commutation (electronic phasing phasing) of two motors on the same axis avoiding the need for mechanical phasing.

The Commutation algorithm results in two modulated reference signals for each instance:

System Variable	Mnemonic	Variable #
Drive-A Motor 1	V_AMotor1	X076
Drive-B Motor 1	V_BMotor1	X077
Drive-A Motor 2	V_AMotor2	X078
Drive-B Motor 2	V_BMotor2	X079

COMMAND: **:Configure:Motor <axis>,<motor>,<mphase>,<A/Bphase> ,<A/Bgain>,<Aoffset>,<Boffset>…**
…,<pos_var>,<torque_var>,<rate_var>,<speed>,<rate_range>

- **<axis>** [NR1] = {1|2|…|n}; where [n] is the maximum number of axes.
- **<motor>** [NR1] {1|2} specifies which motor commutation module is addressed.
- **<mphase>** [NR2] Provides the means to adjust the phase of the modulation of the current/torque command. <mphase> is scaled in the same engineering units as position; but, it is normalized to one electrical cycle of the motor by the <speed> parameter. Proper phasing is required for optimal torque production.
- **<A/Bphase>** [NR2] Sets the phase of the Bmotor output relative to the Amotor. This value in nominally set for 1/3 of a commutation cycle for 3 phase motors and is scaled the same as <mphase>. <A/Bphase> can be trimmed to compensate for quadrature in non-ideal motors.
- **<A/Bgain>** [NR2] Provides compensation for gain differences (Back EMF) of the motor windings. The niminal value of <A/Bgain> is 1.
- **<Aoffset>** [NR2] {scaled in volts} Provides control of the DC bias on the Amotor output. This parameter can be used to minimize torque disturbance.
- **<Boffset>** [NR2] {scaled in volts} Provides control of the DC bias on the Bmotor output. This parameter can be used to minimize torque disturbance.
- **<pos_var>** [NR1] Position variable used for commutation (flux vector rotation).
- **<torque_var>** [NR1] Servo error variable proportional to the desired motor torque.

- **<rate_var>** [NR1] System variable corresponding to the rate of the axis.
- **<speed>** [NR1] Specifies the pole/2 count for the motor.
- **<rate_range>** [NR2] {scaled in rate units} Defines the range of rate that corresponds to the input range of the Phase Advance LUT function.

EXAMPLE: :Config:motor 2,1,-7.3,120,1.012,0.001,-0.002,2107,2164,2083,12,10000

Motor 1 on axis two is a 24 pole motor (12 commutation cycles per revolution). The commutation phase is rotated by –7.3 electrical degrees of the motor for optimum torque production, and the second winding is set for 3 phase commutation; <A/Bphase> = 120 electrical degrees of a commutation cycle. The relative gain between the two phases is non-dimensional, and the current loop offsets are expressed in volts.

EXAMPLE: :Config:motor 2,1,-15.5

This is all that is required to adjust the commutation phase of the motor. The other arguments will remain unchanged.

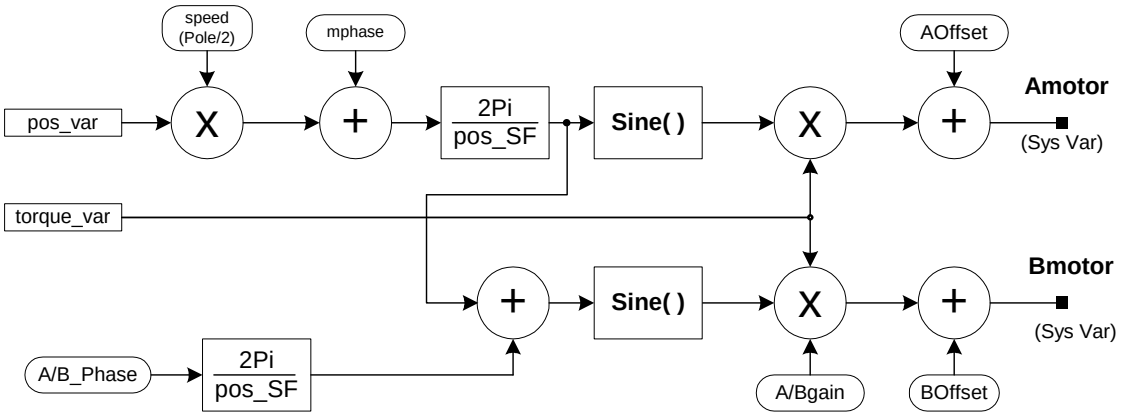
QUERY: :Configure:Motor? <axis>,<motor>

→<mphase>,<A/Bphase> ,<A/Bgain>,<Aoffset>,<Boffset>…

…,<pos_var>,<torque_var>,<rate_var>,<speed>,<rate_range>

EXAMPLE: :Config:motor? 2,1

→ -15.5,120,1.012,0.001,-0.002,2107,2164,2083,12,10000



4.2.17 CONFIGURE:OPTICAL[?]

This command is used to configure the operation of the incremental position accumulator in the AIM hardware, which provides a 16-bit register with the raw optical encoder position. This register is read at the Acutrol frame rate and is further accumulated in software to produce a correlated position over the full scale range of position.

COMMAND: : Configure:Optical <axis>,<enable>,<count_dir>,<count_mode>, <home_sense>,<home>

- **<axis>** [NR1] specifies which axis is being configured.
- **<enable>** {enum} [Enable | Disable] Enables reading the raw encoder position from the AIM.
- **<count_dir>** {enum} [Up | Down] Specifies the direction that the incremental accumulator counts in the AquadB encode mode.
- **<count_mode>** - [AquadB | Pulse] Specifies the accumulation mode of the hardware; AquadB is the default mode. When **Pulse**(/Direction) mode is selected, the encoder "A" input becomes the **Direction** input, and the encoder "B" input functions as the **Pulse**.
- **<home_sense>** {enum} [Hightolow | Lowtohigh] This argument specifies the active edge of the Home (reference) pulse that is used establish the absolute position zero.
- **<home>** [NR1] Specifies the homing algorithm to use (Default = 0 ➔ NONE). 1-15 specifies one of 15 built-in homing sequences. ***Currently there are no built-in home sequences and the homing process is accomplished using ACL script macros, or manually using ACL commands and queries.***

EXAMPLE: :Configure:Optical 1, Enable, Up, Aquadb, Lowtohigh, 0

EXAMPLE: :Con:opt 1, Disable (Disables the optical encoder position update)

QUERY: :Configure:Optical? 1

➔ Disable, Up, Aquadb, Lowtohigh, 0

4.2.18 CONFIGURE:OPTICAL:HOME

Incremental optical encoders need to be initialized to establish an absolute position measurement. This Command is used to enable capturing the optical encoder Reference Position that is used to set the absolute Encoding Position relative to a Home or Reference signal generated by the optical encoder.

After sending this command, the RT computer issues a strobe to bit-2 of the Strobe Register on the corresponding AIM. This strobe resets the E_REF_LATCH status (AIM Status1 Bit-14) and enables the latching of the raw Optical Encoder position at the next occurrence of the optical encoder Reference Pulse. When new data is latched, the RT encoding software performs an initialization sequence. This process is described in more detail in the optical encoder section of TM-9377 Position Transducer Integration Procedure.

COMMAND: :Configure:Optical:Home <Axis>

- **<axis>** [NR1] = { 1 | 2 |….| n }; where [n] is the maximum number of axes. "All" is not a legal option for this command.

EXAMPLE: :Config:Optical:Home 2

COMMAND: :Configure:Optical:Home <Axis>

➔ <Home Status>

- **<axis>** [NR1] = { 1 | 2 |….| n }; where [n] is the maximum number of axes. "All" is not a legal option for this command.
- **<Home Status>** [enum] {Waiting | Done} Reports the R_REF_LATCH statusso that the state of the initialization can be poled by an external ACL process.

EXAMPLE: :Config:Optical:Home? 1

➔ Done

4.2.19 CONFIGURE:OSCILLATOR[?]

This command is used to customize the behavior of the built in oscillator, which generates the sinusoidal reference demand vector in Synthesis profile mode and in other profiling modes whenever the Synthesis demand summing is enabled for this source.

The oscillator Amplitude, Frequency, and Phase are specified in the Demand:Oscillator command in section 4.3.4. One purpose of this command is to enable the Oscillator motion demand vector to be summed with other demand sources while in Position, Absolute Rate, or Track profiling modes. The other configuration parameters set the slew rates and slew modes for changes to the Amplitude and Frequency demands. The amplitude of the sinusoid motion demands change linearly between amplitude set points. The frequency can be configured to slew linear or logarithmic. As the frequency slews, the amplitude of the regulated state is maintained constant. The regulated state may be configured as position, rate, or acceleration.

COMMAND: :Configure:Oscillator <axis>,<Enable|Disable>,<amp_slew>,<freq_slew>,<amp_mode>,<freq_mode>

- **<Enable|Disable>** [enum] {Enable | Disable} Enable (disables) the oscillator.
- **<amp_slew>** [NR2] Defines the linear slew rate from the current Amplitude to the new demand. The slew rate is specified in EU/sec where the engineering units are defined by the <amp_mode> configuration; e.g. (Position - deg/sec), (Rate – deg/s /sec).
- **<freq_slew>** [NR2] Defines the slew rate from the current frequency to the new demand frequency. If <freq_mode> is linear, then slew is in Hz/sec; logarithmic is decades/sec. If in LOG mode the minimum frequency is 0.001 Hz.
- **<amp_mode>** {Pos|Rate|Accel}Defines which motion state is set by the Amplitude demand; the same motion state will be maintained constant as the frequency is changed.
- **<freq_mode>** {Log|Lin} Specifies the frequency sweep mode.

EXAMPLE: :Con:Osc 2,Disab,2.000000,0.100000,Pos,Log

QUERY: :Configure:Oscillator?<axis>

→ <Enable|Disable >,<amp_slew>,<freq_slew>,<amp_mode>,<freq_mode>

Example: :Con:Osc 1

→ Enable, 5.000000, 0.100000, Acceleration, Logarithmic

4.2.20 CONFIGURE:PLANT [?]

The Plant block is used to normalize loop gains such that the compensation for both Observer and Control can be generically set. The physical constants of the real plant are entered as accurately as possible, to ensure precise estimate of observed states and to achieve the desired closed loop response.

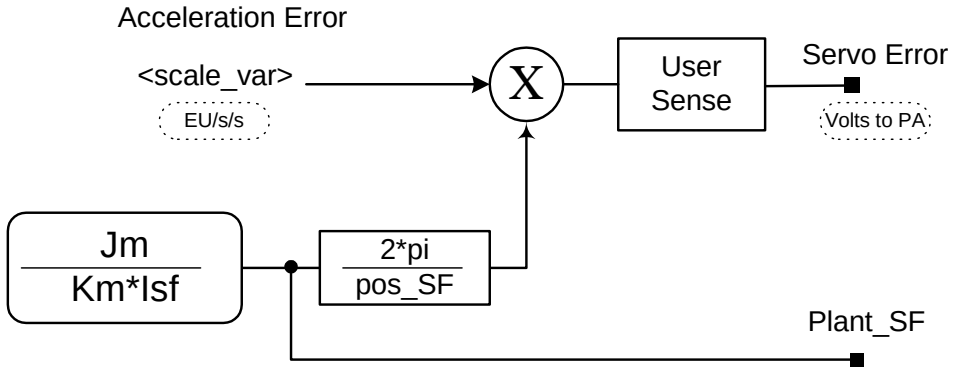
The output of this block generates two System Variables:

System Variable	Mnemonic	Variable #
Normalized Servo Error	V_SERVO_ERR	x164
Plant Scale Factor (gain)	V_NORMEXTGAIN	x219

COMMAND: **:Configure:Plant <axis>,<km>,<Jm>,<Isf>,<in_var>**

- **<axis>** [NR1] = {1|2|…|n}; where [n] is the maximum number of axes.
- **<Km>** [NRf] Torque (force) motor (actuator) constant expressed in engineering units.
- <Jm> [NRf] Inertia or mass constant expressed in compatible engineering units.
- **<ISF>** [NRf] This is the current scale factor of the drive amplifier usually specified in Amps/Volt.
- **<in_var>** [NRf] Specifies the System variable corresponding to the compensated servo error.

EXAMPLE: :Config:Plant 1,0.545400,3.300000,1.500000,1045



Note: User Sense is defined in Configure:Ratiometric(user) <sense> section 4.2.21.

The parameter <pos_SF> is defined in Configure:Scaling section 4.2.26.

QUERY: **:Config:Plant? <axis>**

→ <km>,<Jm>,<Isf>,<in_var>

EXAMPLE: :Config:Plant 1

→ 5.4540000000000E-01, 3.3000000000000E+00, 1.5000000000000E+00, 1045

Assuming English units, the Motor constant Km = 0.54 ft-Lb/Amp, the inertia Jm = 3.3 ft-lb-sec^2, and the power amp scale factor Isf = 1.5 Amp/Volt. The theoretical peak acceleration is computed as follows:

ACCELp = 10(volts) * pos_SF * Km * Isf / Jm / 2Pi

Where pos_SF is the position scale factor (usually 360 degreed/revolution).

For the example above, the peak acceleration is computed as 142 deg/sec^2.

4.2.21 CONFIGURE:RATIOMETRIC[?]

The Ratiometric command is used to convert sinusoidal or linear ratiometric signals to a corresponding position measurement proportional to shaft angle or linear displacement. Typical sinusoid carrier devices are Inductosyns® and resolvers. DC sinusoid devices such as analog optical or demodulated capacitance transducers can also be used. Linear Voltage Differential transducers can be normalized relative to the excitation signal using the divide mode. An offset parameter is provided to effectively rotate or move the transducer null position.

This command includes the setup of the corresponding 10 kHz drive reference carrier signal when carrier modulated devices are employed.

The Ratiometric Encoding module generates position variables, which are scaled such that one (feedback) cycle of the measurement sensor has a range of ±0.5.

System Variable	Mnemonic	Variable #
Fine Raw Position	V_FINE_ANG	x108
Fine Sqrt(Sin^2+Cos^2)	V_FINE_MAG	x106
Fine (Sin – Cos)	V_FINE_DIF	x105
Coarse Raw Position	V_COARSE_ANG	x085
Coarse Sqrt(Sin^2+Cos^2)	V_COARSE_MAG	x206
Coarse (Sin – Cos)	V_COARSE_DIF	x205

Sqrt(Sin^2+Cos^2) is scaled in volts; the theoretical maximum is 10Volts

Sin-Cos is also scaled in volts with a theoretical maximum of 10Volts.

COMMAND: **:Configure:Ratiometric <axis>,<coarse|fine>,<coupling>,<fbk_loss> …**
…,<drv_amp>,<drv_phase>,<mode>

- **<axis>** [NR1] specifies which axis which is being configured.
- **<coarse|fine>** [enum] {Coarse | Fine }specifies which instance of the ratiometric encoder is being referenced.
- **<couple>** [NR2] {nominal value = 0.0±0.1} Sets the cross coupling gain between the feedback channels to compensate for quadrature error in the position transducer.
- **<fbk_loss>** [NR2] {volts} sets the threshold for testing the loss of feedback. If Sqrt(Sin^2+Cos^2) falls below <fbk_loss> then a fault is reported in the ACP status.
- **<drv_amp>** [NR2] Sets the amplitude of the 10 kHz drive reference carrier signal. This argument is scaled as a percentage of full scale drive amplitude.
- **<drv_phase>** [NR2] Sets the phase of the carrier such that the feedback signals have the proper phase for demodulation. This argument is scaled in degrees and has a modulo range of 360° and a resolution of 0.45°.

☞ **<mode>** [enum] {ArcTan2 | Ratio | Passthru | None} defines the computation method used to linearize the ratiometric measurement.

"ArcTan2" => ArcTan2(Fbk1,Fbk2) function used to generate output.

"Ratio" => Division of Fbk1/Fbk2 used to generate output.

"Passthru" => Pass Fbk1 to the output w/o modification.

"None" => Not enabled; output is set to zero.

EXAMPLE: :Config:Ratio 1,Fine,0.022,0.00071,6,59606,20399,A

On axis 1, set the fine coupling calibration parameter; set the drive amplitude and phase, and configure for ArcTan2 ratiometric linearization. Finally the output is offset by 0.022 and the feedback loss is set at 6 volts.

:Config:Rat 2,c,115.5,-0.0013

On axis2, set the coarse coupling calibration gain to –0.0013 and offset the coarse position by 115.5.

QUERY: :Configure:Ratiometric? <axis>,<coarse|fine>

→ <couple>,<fbk_loss>,<drv_amp>,<drv_phase>,<mode>< END>

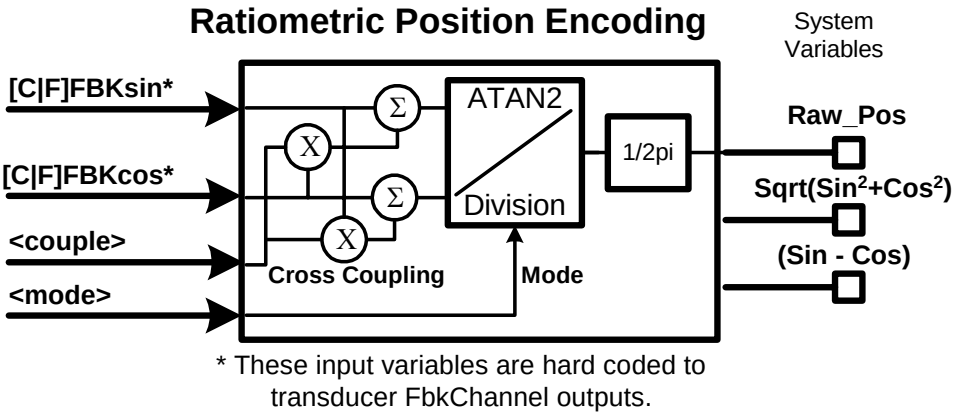
EXAMPLE: Config:rat? 2,c

→ -0.0013,59288,20272,ArcTan2

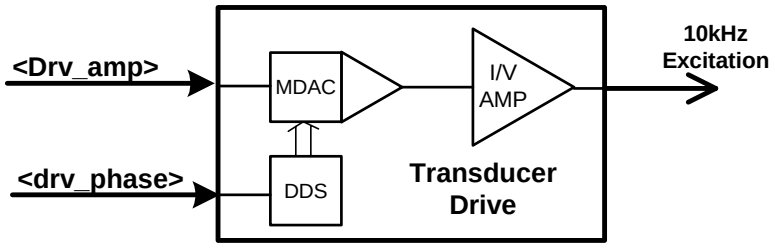
EXAMPLE :Con:Rat? 3,coarse

➔1.2000000000000E-02, 5.0000000000000E+00, 32.263200, 132.556400, Arctan2

Ratiometric Position Encoding and 10KHz Sine Wave excitation block diagrams.



* These input variables are hard coded to transducer FbkChannel outputs.



4.2.22 CONFIGURE:REALTIME[?]

This group of commands is used to configure the software options for real-time data processing. The commands apply generically to all axes. This document does not include the hardware specific setup of real-time interfaces.

For a real-time interface to operate, the Acutrol must be commanded "On-Line". The ACL command to go On-Line is issued from the Acutrol GUI, a remote interface (such as the IEEE488), or the ACL channel of a reflective memory interface. It is recommended when using the VMIC or SCRAMNet+ RT interface options, that the Host computers use the ACL channel of the RM for non-real-time communication/control.

The root Configure:RealTime command is used to request that Acutrol go to the On-Line or Off-Line state.

COMMAND: :CONFig:RealTime <state>

- **<state>** [enum] {Online | Offline} Requests realtime interface to be put On-line or to be taken Off-line.

EXAMPLE: :CONF:Real Online

A query will return the current state of the Acutrol3000 real-time interface.

QUERY: :Config:RealTime?

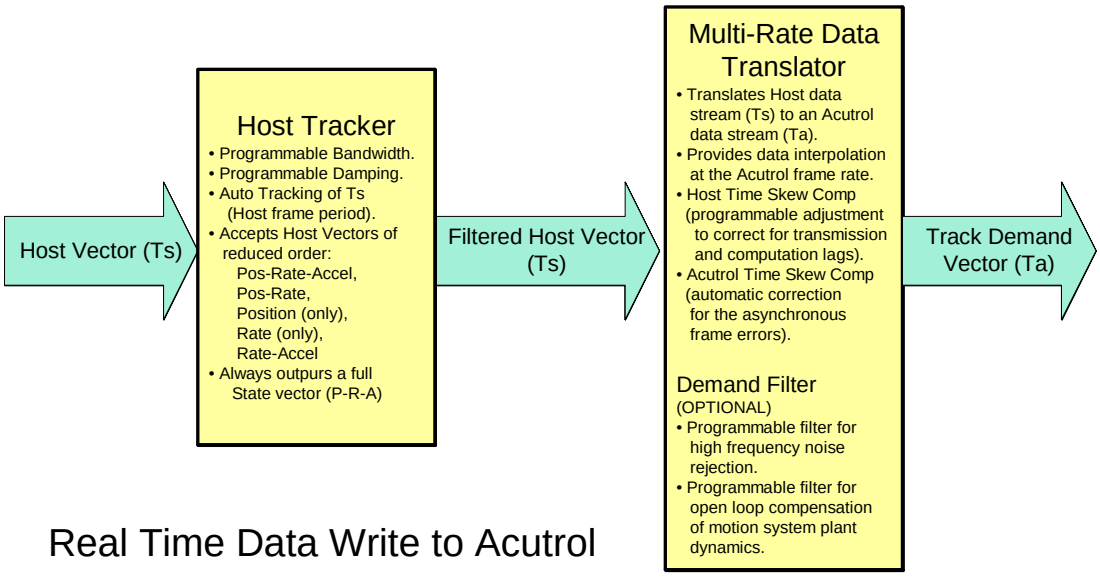
→ <state>

- **<state>** [enum] {Online | Offline | Init | Error} The Query response reports the actual state/success of the On/Off-Line request. The <Init> state is an intermediate state during interface initialization of the RT interface. If the On/Off-Line request could not be achieved, then an <Error> is reported.

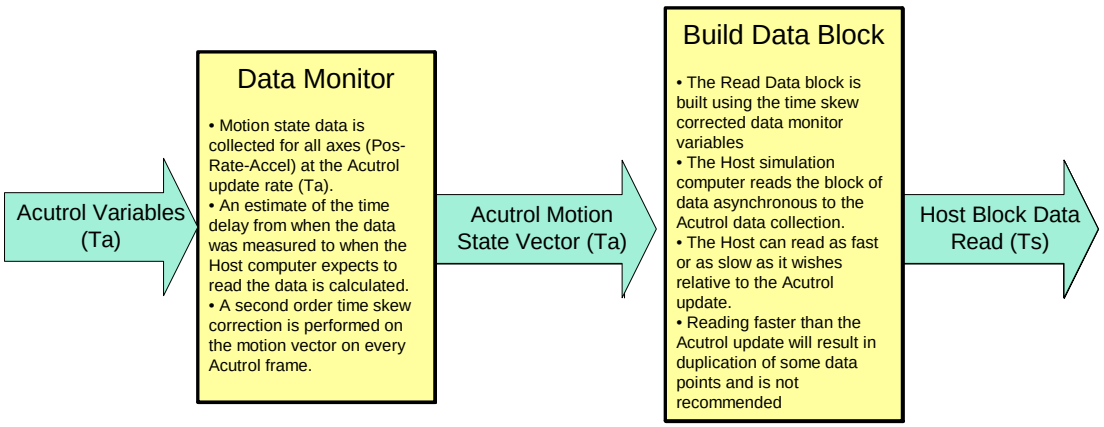
EXAMPLE: :CONF:REAL?

→ Online

The following diagram defines the real-time functional blocks that are configured by Config:RealTime ACL command sub-set.



Real Time Data Write to Acutrol



Real Time Data Read from Acutrol

4.2.22.1 CONFIGURE :REALTIME:BLOCKRECORD[?]

The BlockRecord command allows a remote computer to alter the data to be demanded and monitored by a Block Mode interface. See TM-TBD, Acutrol3000 Variable Definitions for a list of variables that may be demanded and monitored. Up to 48 variables may be demanded or monitored.

COMMAND: :CONFig:RealTime:BlockRecord <direction>,< variable>[,<variable>…]

- <**direction**> [enum] {Demand | Monitor} defines the command reference to either the Demand or Monitor block. Demand applies to Host data sent to Acutrol. Monitor applies to data produced in the Acutrol and read by the Host computer.
- <**variable**> [NR1] is the normal composite System Variable number; where the axis "sub-system" supplying the variable is MOD_{1000}(<variable>), and the specific variable is REM_{1000}(<variable>).

EXAMPLE: :Config:RealTime:Block Monitor, 1560,1561,1562, 2561, 2562, 2563, 3560, 3561, 3562, 102

:Config:Real:Block Demand, 1500, 1501, 1502, 2500, 2501, 2502, 3500, 3501, 3502, 100

A query of the BlockRecord will return the variables currently selected.

QUERY: :Config:RealTime:BlockRecord? <direction>

→ <variable>[,<variable>…]

EXAMPLE: :CONF:REAL:BLOC? DEM

→ 1082, 1083, 1176, 2082, 2083, 2176, 3082, 3083, 3176, 100

:Config:RealTime:Block? Mon

→ 1560, 1561, 1562, 2561, 2562, 2563, 3560, 3561, 3562, 102

For a detailed explaination of the configuration and operation of real-time interfaces and/or a description of the real-time variables in the examples above, refer to the Acutronic manual TR-9374A Acutrol3000 Real-time Data Communication Manual. A limited description of the real-time processes are provided in the **:Configure:RealTime** sections that follow.

4.2.22.2 CONFIGURE:REALTIME:MONITOR[?]

This command is used to configure the real-time monitor routine that acquires the specified motion data, performs skew correction/prediction of the data, and makes a new set of motion variables available to the real-time interfaces. Monitor data is either read by the Host computer (reflective memory interface) or is sent by Acutrol (parallel interface).

The following System variables are produced as a result of time skew correction.

System Variable	Mnemonic	Variable #
Corrected Position variable	V_FHOST_POS_MON	X560
Corrected Rate variable	V_FHOST_RATE_MON	X561
Corrected Acceleration variable	V_FHOST_ACCEL_MON	X562
Corrected Jerk variable	V_FHOST_JERK_MON	X563

COMMAND: :Configure:RealTime:Monitor <axis>,<pos_var>,<rate_var>,<accel_var>

- <**(motion state)_var**> [NR1] is a normal composite System variable number which references a motion state to be corrected. All specified motion variables must be of the same origin (i.e all raw or all predicted.) to ensure data coherence.

Note: This process begins when the interface goes online. The frame time is inferred from the host update rate.

EXAMPLE: :Config:RealTime:Monitor 1,1082,1083,1176

:Config:RealTime:Monitor 2, 2082, 2083, 2176;Mon 3, 3082, 3083, 3176

QUERY: :Configure:RealTime:Monitor? <axis>

➔ <pos_var>,<rate_var>,<accel_var>

EXAMPLE: :Config:RealTime:Monitor? 1

➔ 1082,1083,1176

:CON:REAL:MON? 1;MON? 2;MON? 3

➔ 1082, 1083, 1176; 2082, 2083, 2176; 3082, 3083, 3176

4.2.22.3 CONFIGURE:REALTIME:SETUP[?]

This command is provided to configure the overall operation of the RT algorithms. Real-time Tracker and Translator modules generally may be bypassed or the simulation specialist may choose one of several algorithms that are provided for each function. The different algorithms produce different results depending on the nature of the RT data structure and transfer protocols.

COMMAND: **:Configure:RealTime:Setup <host_period>,<track mode>,<Demand translator mode>,<MonitorTranslator mode>, <Monitor Protocol>, <demand_delay>, <monitor_delay>, <data format>, <hostTimeout>, <host_safety_buffer>**

- **<host_period>** [NR2] {µseconds} User programmable estimate of the host computer simulation period.
- **<track mode>** [Disabled | 1 | 2] Defines and/or enables the algorithm used for motion state tracking.
  Disabled: No tracking applied, output = input. This is only valid in PRA track mode
  1: PRA integration with PR feedback loops and no PR input delay
  2: PRA integration with PR feedback loops and PR input delay
  3: PRA integration with no PR feedback loops
- **<Demand Translator mode>** [Disabled | 1 | 2] Defines and/or enables the algorithm used for demand time skew correction.
  Disabled: No translation, output = input
  1: Multi-rate State Synchronization with PRA input delay and PR feedback loops
  2: Multi-rate State Synchronization with PRA input filtering and PR feedback loops
  3: Multi-rate State Synchronization with no PRA input delay and no PR feedback loops
- **<Monitor Translator mode>** [Disabled | 1 | 2] Defines and/or enables the algorithm used for monitor skew translation .
  Same options as <Demand Translate mode>.
- **<Monitor protocol>** [Disabled | ACT2000 | DRNHS] Defines and/or enables the algorithm used for realtime monitor communication .
  Disabled: No monitor variables are output
  ACT2000: ACT2000 compatible
  DRNHS: Host direct read with no handshaking (uses "keep out" method)
- **<Demand_delay>** [NR2] {µseconds} User defined programmable delay that is used to adjust the reference point for time skew correction of the Host Demand motion states. Limits are ±3 Host frames entered in µseconds.
- **<monitor_delay>** [NR2] {µseconds} User defined programmable delay that is used to adjust the reference point for time skew correction of the Host Monitor motion data. Limits are ±3 Host frames entered in µseconds.
- **<data format>** [Binary | Float | Double] - "Binary" means all realtime data (demand and monitor) is 32-bit signed integer. "Float" means 32-bit IEEE floating point. "Double" means 64-bit IEEE floating point.
- **<hostTimeout>** [NR2] **{** µseconds **}** If host does not send another demand update within this time period (uSec), generate a soft abort. If hostTimeout is 0.0, then the timeout is disabled.
- **<monitorProtocol>** [Disabled | 1 | 2] Defines the monitor communications protocol. 0 = A2K compatible, 1 – Host direct read with no handshaking**.**

☞ **<host_safety_buffer>** [NR2] {µseconds} This argument defines the (time) window in which the Host computer must read the monitor data to avoid the generation of a RT communication read error.

EXAMPLE: :Con:Realtime:Setup 2000, Disabled, 4, 5, ACT2000, 0.0, 0.0, Double, 0.0, 1000.

QUERY: :Configure:RealTime:Setup?

➔ <host_period>,<track mode>,<Demand translator mode>,<MonitorTranslator mode>, <Monitor Protocol>, <demand_delay>, <monitor_delay>, <data format>, <hostTimeout>, <host_safety_buffer>

EXAMPLE: :Con:Realtime:Setup?

➔ 2000.000000, Disabled, 4, 5, Act2000, 0.000000, 0.000000, Double, 0.000000, 1000.000000

4.2.22.4 CONFIGURE:REALTIME:TRACKER[?]

The Host Tracker is a real time module that accepts real-time host demands and outputs a 'full' Demand State Vector consisting of position, rate, and acceleration states. Ideally the input includes all three motion states; but often, one or two states are missing. The tracker runs at the frame rate of the host computer and is synchronized by the actual data transfer. One instance of the Tracker runs for all axes. The dynamic response of the Tracker can be optimized for each axis.

The Tracker System variables are defined as follows:

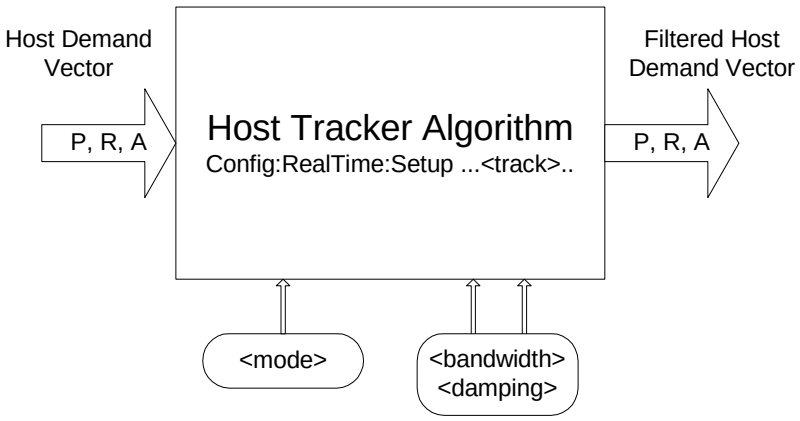
System Variable	Mnemonic	Variable #
Host Position Demand Variable	V_HOST_POS_DMD	x501
Host Rate Demand Variable	V_HOST_RATE_DMD	x502
Host Acceleration Demand Variable	V_HOST_ACCEL_DMD	x503
Host Jerk DemandVariable	V_HOST_JERK_DMD	x504
Filtered Host Position Demand Variable	V_FHOST_DMD_POS	x284
Filtered Host Rate Demand Variable	V_FHOST_DMD_RATE	x285
Filtered Host Acceleration Demand Variable	V_FHOST_DMD_ACCEL	x286
Host Filtered Jerk Demand Variable	V_FHOST_DMD_JERK	x287

The RT algorithms currently do not use the variables for Jerk demands; they are shown here as place keepers for future algorithms.

Command: :Configure:RealTime:Tracker <axis>,<mode>,<bandwidth>,<damping>

- <**axis**> [NR1] = {1|2|…|n}; where [n] is the maximum number of axes.
- <**mode**> [enum] {Disabled | PRA | PR | P | RA | R} defines which real-time demand states are input to the Host Tracker. The <**mode**> argument also defines the filter topology required to produce a 'full' Demand State vector. "Disabled" implies that the Demand data passes unaltered to the output variables. For a detailed explaination of the Tracker Modes, refer to TR-9374A Acutrol3000 Real-time Data Communication Manual.
- <**bandwidth**> [NR2] defines the un-damped natural frequency Wn of the tracker.
- <**damping**> [NR2] defines the damping factor (zeta) of the tracker.

EXAMPLE: : Configure:RealTime:Tracker 1,PRA,80,0.707



4.2.22.5 CONFIGURE:REALTIME:TRANSLATE[?]

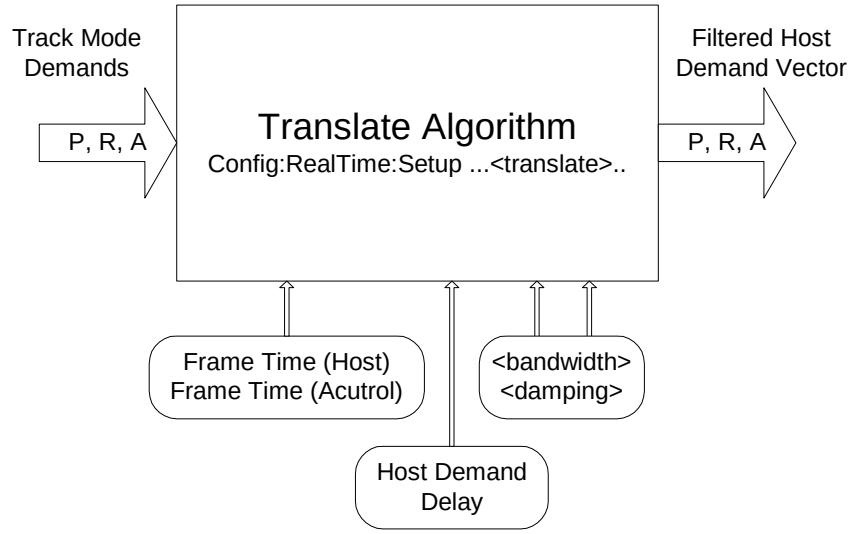
The Translation routine runs at the Acutrol3000 frame rate. The Host demand data is generally updated at a rate less than the Acutrol frame rate, and in fact may be asynchronous. The function of the Translator is to transform vectored data from the Host to the Acutrol frames using extrapolation and time skew correction techniques.

The inputs to the Translate block are the output variables of the Host Tracker defined in section 4.2.22.4.

System Variable	Mnemonic	Variable #
Track Demand Position Variable	V_TRACK_POS_DMD	X540
Track Demand Rate Variable	V_TRACK_RATE_DMD	X541
Track Demand Acceleration Variable	V_TRACK_ACCEL_DMD	X542
Track Demand Jerk Variable	V_TRACK_JERK_DMD	X543

COMMAND: :Configure:RealTime:Translator <axis>,<direction>,<bandwidth>,<damping>

- <**axis**> [NR1] = {1|2|…|n}; where [n] is the maximum number of axes.
- <**direction**> [enum] {Demand | Monitor} defines the command reference to either the Demand or Monitor translator. Demand applies to Host data sent to Acutrol. Monitor applies to data produced in the Acutrol and read by the Host computer.
- <**bandwidth**> [NR2] defines the un-damped natural frequency Wn of the optional Translator low pass filter.
- <**damping**> [NR2] defines the damping factor (zeta) of the optional Translator low pass filter.



4.2.23 CONFIGURE:RESTORE[?]

This command

COMMAND: :Configure:Restore <axis |ALL>,<fileversion>

- **<axis |ALL>** Specifies the axis that will have it's configuration data restored.
- **<fileversion>** NR1 {001…999} If <fileversion> is not specified, this command restores the system configuration from the default XML file (config.001.xml). If <fileversion> is specified, this command restores configuration data from the file named **config.<fileversion>.XML,** for the axis specified. If no arguments are specified, then all axes are restored from the default <fileversion>.

Query: :Configure:Restore? <Axis | All>

Responds with currently loaded XML <fileversion> and the comment that was defined during the last save.

Example: :Config:Restore?

➔5,"As Shipped configuration 5/5/05"

4.2.24 CONFIGURE:RESTORE:GROUP

This system feature is not implemented at this time.

COMMAND: :Configure:Restore:Group <axis |ALL>, <group>[,<fileversion>]

This commands loads the system with the configuration information for a specified group from a specific configuration file. If the file version is not specified, then the most resent referenced file version is used.

<group> is one of {User, System, Interface, Transducer, Servo}

The groups indicate which subset configuration command data is referenced. The grouping of configuration date is defined here:

- User: Analog, Event, GPIO(6-7), Oscillator, limit(other than absolute)
- System: Aim, Analog, Axis, Discrete, GPIO(0-5), Scaling, Variable, interlock, limit absolute
- Interface: Motion States, real time, interface
- Transducer: correlate, feedback channel, Optical, Ratiometric
- Servo: demand sum, filter, motor, plant, summer.

When this feature is implemented a detailed list for each group will be defined and inserted into this section.

4.2.25 CONFIGURE:SAVE[?]

COMMAND: :Configure:Save [<axis |ALL>[,<fileVersion>, <comment>]]

If fileVersion is not specified, saves in the default XML file number (001). If fileVersion is specified, saved in file <fileversion>.

Comment – ascii text (up to 72 chars) saved with the file and displayed with the listing.

EXAMPLE: :Configure:Save ALL,100,"Default Configuration for Inventory"

(Saves all axes to the XML Configuration File version100)

:Configure:Save 2,100,"Default Configuration for Inventory"

(Saves only Axis 2 to the XML Configuration File version100)

QUERY: Configure:Save?

➔ <number of Config Files>,<config File#>,<"File# Comment">[,<config File#>,<"File# Comment">…]

Query generates a listing of all existing configuration files.

EXAMPLE: :Config:Save?

➔ 6, 1, "This is Ver.001", 2, "This is Ver.002", 3, "This is Ver.003", 4, "This is Ver.004", 10, "GLITCH TRACKING", 100, "Default Configuration for Inventory"

4.2.25.1 CONFIGURE:SAVE:GROUP

This system feature is not implemented at this time.

The save group command is used to save subsets of the configuration data. It only updates a subset of the configuration data. The save group command may not be used to save to a new file. It may only update an existing file. Groups are described with the **Configure:Restore:Group** command.

COMMAND: :Configure:Save:Group <axis |ALL>, <group> [,<fileVersion>[,<comment>]]

- **<group>** is one of {User, System, Interface, Transducer, Servo}
- **<fileVersion>** is an integer between 0, 999 inclusive. If the file version does not exists, an error will be returned.

4.2.26 CONFIGURE:SCALING[?]

This command defines the demand scaling for all motion states based on the measurement range of position. This range usually corresponds to one revolution for rotary systems, and a measurement range somewhat greater than the physical range for limited motion systems. The engineering units are specified for position, and the other states are successive time derivative units (e.g. position – degrees, rate – deg/s, accel – deg/s/s). The full-scale range is defined with the same units/scaling as position and defines the position range before the position measurement rolls over. In reality, the full-scale range must have an integer common factor with the base measurement range of the position encoding system. The format arguments define the default output format whenever a variable is specified to use demand scaled properties. See section 4.2.28 (:Config:Variable) for a description of demand scaling.

COMMAND: :Configure:Scaling <axis>,<FS_pos>,<pos_SF>,<pos_offset>,<eng_unit>…
…[,<pos_format>[,<rate_format> [,<acc_format>[,<jerk_format>[,<volt_format>]]]]]

- **<axis>** [NRf]/[NR1] = {1|2|…|n |ALL}; where [n] is the maximum number of axes. "ALL" means that the System:Scaling command applies to all axes defined in the system.
- **<FS_pos>** [NFf] is the full scale position range in <eng_units>. Default is 360.
- **<pos_SF>** [NRf]/[NR2] This is the position scale factor, which transforms the raw position measurements to motion state variables that are scaled in the user specified engineering units. The scale factor must be defined consistently with the <engineering units> described below. If a variable has a fixed dynamic range in the native internal scaling (±0.5), then the range of the variable in the user engineering units is equal to the internal range times the <scalefactor>. Default is 360 (±180).
- **<pos_offset>** [NRf]/[NR2] The offset permits biasing the input/output data relative to the internally number convention. This feature can be used to implement unipolar command ranges. The offset must be scaled consistently with the engineering units defined below. Default is 0.
- **<eng_unit>** {string – 12 characters maximum} Defines the text used to display engineering units. Default is deg.
- **<pos_format>** [enum] { NR1 | NR2 | NR3 | H8 | H16 | H32 | H64 | B16 | B32 | B64} Defines the output format for position demand scaled response data.
- **<rate_format>** [enum] { NR1 | NR2 | NR3 | H8 | H16 | H32 | H64 | B16 | B32 | B64} Defines the output format for rate demand scaled response data.
- **<acc_format>** [enum] { NR1 | NR2 | NR3 | H8 | H16 | H32 | H64 | B16 | B32 | B64} Defines the output format for acceleration demand scaled response data.
- **<jerk_format>**[enum] { NR1 | NR2 | NR3 | H8 | H16 | H32 | H64 | B16 | B32 | B64} Defines the output format for jerk demand scaled response data.
- **<volt_format>** [enum] { NR1 | NR2 | NR3 | H8 | H16 | H32 | H64 | B16 | B32 | B64} Defines the output format for position demand scaled response data.

The Enumerated arguments must be entered exactly as defined.

4.2.27 CONFIGURE:SUMMER[?]

Summers are used in the servo structure to numerically add 1, 2, or 3 signals (System Variables) together. The result is a new variable that is available to other processes. By configuring the inputs to the summere, the servo topology is defined. In the standard implementation of filters and summers, a Summer(n) output is automatically routed to a corresponding Filter(n) input. Summers (and filters) are sequentially executed in two blocks. The first (in time) is the Observer block consisting of five summers/filters. The second block is executed at a later time after all controller inputs are collected, and consists of 10 summer/filters used to implement the main motion servo loops. The third block consists of five (5) filters that are executed after the plant gains are normalized; these are used with pressure, torque, or acceleration sensors to close an inner servo loop. Summer and filter blocks are computed in sequence from lower to higher summer/filter number.

COMMAND: **:Configure:Summer <axis>,<summer#>,<mod_flag>,<in_var1>,<gain1>,<in_var2>,<gain2>…**
…,<in_var3>,<gain3>

- **<axis>** [NR1] = { 1 | 2 |….| n }; where [n] is the maximum number of axes.
- **<summer#>** [NR1] = { 1 | 2|….| 15 }; controller summer numbers.
- **<summer#>** [NR1] = { 20 | 21 |…|24 }; observer summer numbers.
- **<mod_flag>** [enum] {Enable | Disable} Specifies that the summer output must be modulo corrected based on the position full scale range <FS_pos> specified in 4.2.26
- **<in_var1, 2, 3>** [NR1] Specifies the System Variable that is connected to each of the three summer inputs.
- **<gain1, 2, 3>** [NR2] Specifies the gain that is applied to each of the three summer inputs. A negative gain can be used to compute a difference.

EXAMPLE: :Config:Sum 1,1,0,1024,1,1022,-1,1000,0

(Axis 1 System Variable 1022 is subtracted from variable 1024; the third input is not used).

Query: **:Configure:Summer? <axis>,<summer>**

→ <mod_flag>,<in_var1>,<gain1>,<in_var2>,<gain2>,<in_var3>,<gain3>

EXAMPLE: :Config:Sum? 1,2

→ 1009,1,1083,-1,1118,0

The Output of each Summer produces a corresponding System Variable and is summarized in the table below:

System Variable	Mnemonic	Variable #
Control Summer1	V_CTRL1_ERR	x030
Control Summer 2	V_CTRL2_ERR	x032
Control Summer 3	V_CTRL3_ERR	x034
Control Summer 4	V_CTRL4_ERR	x036
Control Summer 5	V_CTRL5_ERR	x038
Control Summer 6	V_CTRL6_ERR	x040
Control Summer 7	V_CTRL7_ERR	x042
Control Summer 8	V_CTRL8_ERR	x044
Control Summer 9	V_CTRL9_ERR	x046
Control Summer 10	V_CTRL10_ERR	x048
Control Summer 11	V_CTRL11_ERR	x050
Control Summer 12	V_CTRL12_ERR	x052
Control Summer13	V_CTRL13_ERR	x054
Control Summer 14	V_CTRL14_ERR	x056
Control Summer 15	V_CTRL15_ERR	x058
Observer Summer 20	V_OBS1_ERR	x060
Observer Summer 21	V_OBS2_ERR	x062
Observer Summer 22	V_OBS3_ERR	x064
Observer Summer 23	V_OBS4_ERR	x066
Observer Summer 24	V_OBS5_ERR	x068

Errors:

1. Invalid <axis> Parameter out of range error
2. Invalid <summer#> Parameter out of range error
3. Invalid <in_var> Parameter out of range error

Note:

1. A summer may have unused inputs; the associated gain should be set to “0.0” and the input variable set to the null variable X118.

4.2.28 CONFIGURE:SYSTEM:RESTORE

COMMAND: :Configure:System:Restore

Restores the **system.xml** file and is provided as a complementary command to the Configure:System:Save command.

WARNING! Use this command at your own risk. There are many initialization dependencies that may or may not get changed when their values are reloaded. The only safe initialization of the system parameters/configuration is performed during the initial boot sequence of the RT Acutrol3000 application.

COMMAND: Configure:System:Restore?

EXAMPLE: :Configure:Restore? All

➔ 100, "Default Configuration for Inventory HSH"

4.2.29 CONFIGURE:SYSTEM:SAVE

COMMAND: :Configure:System:Save <comment>

Saves system.xml file

Comment – ascii text (up to 72 chars) saved with the file and displayed with the listing.

Note: Multiple versions of the System configuration are <u>not</u> supported.

4.2.30 CONFIGURE:VARIABLE[?]

This command is used to configure each System variable that is defined in the Acutrol3000 controller. A variable name and scaling method <state> must be specified as a minimum.

Currently Demand scaling is not implemented; therefore, all variables must be individually and explicitly configured and the **<state>** parameter must be set to "None".

COMMAND: :Configure:Variable < var_num>,<var_name | Factory>,<state>…

…,<scalefactor>,<offset>,<output_format>,<engineering_units>

- **<var_num>** [NR1] {1..x999} specifies the variable being modified; x => axis.
- **<var_name>** [string – 26 characters max] User defined nomenclature which identifies the variable for pick box and display in user interfaces. Specifying Factory resets the default (factory defined) name for this variable.
- **<state>** [enum] {None | Position | Acceleration | Rate | Jerk | Volts} Specifies the motion state if Demand Scaling is being defined. If "None" is specified, then the optional arguments defined below must be included.
- **<scalefactor>** [NRf] The scale factor transforms input/output data to/from the native scaling that is used internally in the controller. The scale factor must be defined consistently with the <engineering units> described below. Note, if the variable has a fixed dynamic range in the native internal scaling, then the range of the variable in the user engineering units is equal to the internal range times the <scalefactor>.
- **<offset>** [NRf]/[NR2] The offset permits biasing the input/output data relative to the internal number convention. This feature can be used to implement unipolar command ranges. The offset must be scaled consistently with the engineering units defined below.
- **<output_format>** [enum]{ NR1 | NR2 | NR3 | H8 | H16 | H32 | H64 | B16 | B32 | B64}
- **<engineering_units>** {string – 12 characters maximum} Defines the text used to display engineering units.

Example: :Config:Variable 1082,"Estimated Position",N,1,0,NR2,"Degrees"

:Config:Variable 1166,"Position Feedback",P (future)

The query command always returns the complete set of arguments even for Demand Scaled variables.

QUERY: :Configure:Variable? < var_num>

→<var_name | Default>,<state>…

…[,<scalefactor>,<offset>,<output_format>,<engineering_units>]

4.3 DEMAND

The DEMand Subsystem commands are used to input motion demand data to the Command Profiler. The data can be static motion demands, real-time state vector motion demands, or parameters for synthesized sinusoidal motion.

Also data can be input to generic registers for application specific purposes.

4.3.1 DEMAND:ACCELERATION[?]

The ACCeleration command is used to send acceleration demand data to the specified axis controller.

COMMAND: :Demand:Acceleration <axis>,<acceleration data>

- **<axis>** [NR1] {1..n} specifies the axis being modified.
- **<acceleration data>** [NRf]

EXAMPLE: :DEM:ACCeleration 2,1000

:D:A 1,100;A 3,450.

:DEM:A 2,#14<4 bytes binary data>

The ACCeleration query returns the current acceleration demand.

QUERY: :Demand:Acceleration? <axis>

→ <acceleration demand data>

EXAMPLE: :DEM:ACC? 1

→ 1000

:DEM:ACC? 1;ACC? 2

→ #14<4 bytes binary axis 1 acceleration demand>;

#14<4 bytes binary axis 2 acceleration demand>

Errors:

1. Invalid <axis>. Set PARAMETER OUT OF RANGE Bit in Error Byte.

4.3.2 DEMAND:DELTAPOSITION[?]

The DELtaposition command is used to increment or decrement the position demand of the specified axis. The value returned by the position demand query :DEMand:POSition? will change after each DELTAposition command to reflect the new demand position.

COMMAND: :Demand:Deltaposition <axis>,<data>

- **<axis>** [NR1] {1..3} specifies the axis being modified.
- **<data>** [NRf] specifies the adjustment to the current position demand as a relative position step.

EXAMPLE: :DEM:Delta 1,-45.0; The current position demand is reduced by 45 'degrees' and the result is re-commanded as the new position.

:D:D 2,#14<4 bytes binary position delta data>

The DELtaposition query will return the last position increment/decrement received for the selected axis.

QUERY: :Demand:Deltaposition? <axis>

→ <position demand delta>

EXAMPLE: :DEM:DEL? 2;DEL? 1

→ 1.00000;-45.00000

:D:D? 3

→ #14<4 bytes binary position delta>

Errors:

1. Invalid <axis>. Set the PARAMETER OUT OF RANGE bit in the Status Byte.

4.3.3 DEMAND:INPUTVARIABLE

The InputVARiable command sends data from the CIC (Controller in Charge) to one of 12 Input Registers. The use of these data are defined by CONFiguration procedures or by custom software; data can be output to analog channels or selectively summed to nodes of the servo controller. The Input Variable Registers are "Supervisor variables" and are used by in the ACP by being mapped to ACP variables or Supervisor Variables have to be implemented in the software as Supervisor sub-system variables. This could be done using "0" as the sub-system designator for the supervisor.

Note the data type and scale factor must be consistent with the utilization of the variable in the system.

COMMAND: **:Demand:Ivariable <input_variable>,<data>**

- **<input_variable>** [NR1] { 1 | 2 |..|12 }
- **<data>** [float]

EXAMPLE: :DEM:IVAR 7,100.8;IVAR 9,#14<4 bytes>

The Input Variable query returns the current value of the selected input register.

QUERY: **:Demand:Ivariable? <input_variable>**

→ <input_variable data>

EXAMPLE: :DEM:IVAR? 7;IVAR? 9

→ 100.8000;-23.450

:D:I? 3

→ #14<4 bytes binary ivariable data>

Errors:

1. Invalid <IVAR>. Set the PARAMETER OUT OF RANGE

4.3.4 DEMAND:OSCILLATOR[?]

This command specifies the peak position displacement, frequency, and initial phase for synthesized sine waves, used in the SYNThesis servo mode.

COMMAND: :Dem:Oscillator <axis>,<magnitude>,<frequency>,<phase>

- <**magnitude**> [NR2] The amplitude of the Sinusoid at steady state; after amplitude slew. The amplitude is proportional to the state (Position, Rate, or Acceleration) depending on the <amp_mode> specified in Configure:Oscillator section 4.2.18.
- <**frequency**> [NR2] Frequency of Sinusoid in Hz.
- <**phase**> [NR2] Initial phase of Sinusoid in Degrees.

EXAMPLE: :DEM:OSC 1,10.0,2.50,90

:D:O 1,#212<4 bytes magnitude data>
<4 bytes frequency data>
<4 bytes phase data>

The OSCillator query returns the currently demanded <magnitude>, <frequency>, and <phase> to be used in SYNThesis mode.

QUERY: **:Demand:Oscillator? <axis>**
→ <magnitude><frequency><phase>

EXAMPLE: :DEM:OSC? 1
→ 10.000;2.5000;90.00000 (ASCII Float Mode).

:D:O? 2
→ #212<4 byte binary magnitude demand data>
<4 byte binary frequency demand data>
<4 byte binary phase demand data> (Binary Mode).

Errors:

1. Invalid <axis>. Set PARAMETER OUT OF RANGE bit.
2. Invalid <magnitude> Set PARAMETER OUT OF RANGE bit.
3. Invalid <frequency > Set PARAMETER OUT OF RANGE bit.
4. Invalid <phase> Set PARAMETER OUT OF RANGE bit.

4.3.5 DEMAND:POSITION[?]

This command is used to send position demand data to the specified axis controller.

COMMAND: **:Demand:Position <axis>,< position >**

EXAMPLE: :DEM:POS 1,135.223

:DEM:POS 1,#14<4 bytes binary data>

The POSition query will return the current position demand for the selected axis.

QUERY: **:Demand:Position? <axis>**

→ <position>

EXAMPLE: :DEM:POS? 2;POS? 1

→ 170.00000;-32.19877

:D:P? 3

→ #14<4 bytes binary position demand>

Errors:

1. Invalid <axis>. Set the PARAMETER OUT OF RANGE bit.

4.3.6 DEMAND:RATE[?]

The RATE command is used to send rate demand data to the specified axis controller.

COMMAND: :Demand:Rate <axis>,<rate>

EXAMPLE: :DEM:RATE 2,100.0
:D:R 1,1.11111;R 3,45.125
:DEM:R 2,#14<4 bytes binary data>

The RATE query will return the current rate demand for the selected axis.

QUERY: :Demand:Rate? <axis>
→ <rate>

EXAMPLE: :DEM:RATE? 2;RATE? 1
→ 3200.0000;-150.1234

:D:R? 3
→ #14<4 bytes binary rate demand>

Errors:

1. Invalid <axis>. Set the PARAMETER OUT OF RANGE bit.

4.3.7 DEMAND:SHORTVECTOR[?]

The Short VECtor command sends the demand states (position and rate) to the specified axis controller.

COMMAND: :Demand:ShortVector <axis>,<position demand>,<rate demand>

EXAMPLE: :DEM:SVEC 4,1.23456,24.6802

:DEM:SVEC 6,#18<8 bytes data>

:D:S 6,#0<8 bytes data>

The Short VECtor query returns the current position and rate demand from the selected axis.

QUERY: :Demand:ShortVector? <axis>

→ [#18]<position demand data><rate demand data>

EXAMPLE: :DEM:SVEC? 2

→ 1.23456;24.6802

:D:S? 3

→ #18<4 bytes position demand><4 bytes rate demand>

Errors:

1. Invalid <axis>. Set the PARAMETER OUT OF RANGE bit.

4.3.8 DEMAND:VECTOR[?]

This command sends three demand states (position, rate, and acceleration) to the specified axis controller.

COMMAND: :Demand:Vector <axis>,<position demand>,<rate demand>,<acceleration demand>

EXAMPLE: :DEM:VECT 2,1.23456,24.6802,135.791

:DEM:VECT 4,#212<12 bytes data>

:D:V 1,#0<12 bytes data>

The VECTor query returns the current position, rate, and acceleration demand from the selected axis.

QUERY: :Demand:Vector? <axis>

→ [#212]<position demand data><rate demand data><acceleration demand data>

EXAMPLE: :DEM:VEC? 2

→ 1.23456;24.6802;135.791

:D:V? 3

→ #212<4 bytes position demand><4 bytes rate demand>…

…<4 bytes acceleration demand>

Errors:

1. Invalid <axis>. Set the PARAMETER OUT OF RANGE bit.

4.3.9 DEMAND:WORLDCOORDS[?]

This feature is not implemented at this time.

This command allows the user to send demands to the motion system in world coordinates. A GCT (Global Coordinate Transform) module must be loaded in Acutrol for these commands to function. If a GCT module is not loaded, the results of these commands are undefined.

All parameters to this command must be specified. The number of parameters is equal to the number of degrees of freedom and is system dependent. See the system's GCT documentation for details.

COMMAND: :Dem:WorldCoords <GCT parameter 1>,<GCT parameter 2>[,…]

EXAMPLE: :DEM:WORld 45.0,90.0,-90.0 Roll, pitch, yaw

:D:W #224<24 bytes GCT data> X, Y, Z, roll, pitch, yaw

The WORld query returns the current GCT demands. The number of parameters returned is equal to the number of degrees of freedom and is system dependent. See the system's GCT documentation for details.

QUERY: :Demand:WorldCoords?

→ [#nNN]<GCT parameter 1>[;]<GCT parameter 2>[;…]

EXAMPLE: :DEM:WOR?

→ 45.00000,90.00000,-90.00000

:D:W?

→ #224<24 bytes GCT demand data>

{Let's table even the definition of this until we have the other issues completed.}

4.4 INTERFACE

4.4.1 INTERFACE:IEEE488 [?]

COMMAND: **:Interface:IEEE488 <board address>,<hsp_dav>**

- **<boardAddress>** [NR1] (0..1)
- **<hsp_dav>** [NR1] {0..1} high speed data available.

4.4.2 INTERFACE:IEEE488:LOCAL

This command sends the IEEE488 interface back to local mode.

COMMAND: **:Interface:IEEE488:Local**

4.4.3 INTERFACE:PARALLEL[?]

COMMAND: :Interface:Parallel <sync_ID>,<read_delay>,<write_delay>,<LO/HI>,<PUL/LEV>,<LPULSE>,<EXCH PINS>

- **<sync_ID>** [NR1]
- **<read_delay>** [NR1] {0..7}
- **<write_delay>** [NR1] {0..7}
- **<LO/HI>** [Hex Digit] Controls the active level of the 4 parallel interface handshake signals. See Table 10/11 of TM-9216A
- **<PUL/LEV>** [Hex Digit] Controls the active handshaking mode of the parallel interface handshake signals. See Table 12/13 of TM-9216A
- **<LPULSE>** [Hex Digit] Controls the data settling mode when leading-edge handshaking is used. See Table 14/15 of TM-9216A
- **<EXCH PINS>** [enum] (True | False) If <EXCH PINS> = Exchange, drive the ACK signal onto the Req pin and vice versa. The REQ and ACK pins exchange functions, allowing direct connection between two identical boards.

Example: :Interface:Parallel 4096, 225, 225,15, 14, 2, FALSE

QUERY: :Interface:Parallel?

→<sync_ID>,<read_delay>,<write_delay>,<LO/HI>,<PUL/LEV>,<LPULSE>,<EXCH PINS>

Example: :Interface:Parallel?

→4096, 225, 225,15, 14, 2, FALSE

4.4.4 INTERFACE:RS232[?]

This command is not implemented in the ACL command set.

Acutrol may be controlled using a serial RS232 interface using a RS232 <=> Ethernet converter. This capablilty is described in TM-xxxx RS232 Interface Manual for the Acutrol3000.

COMMAND: **:Interface:RS232 <port>,<baud>,<parity>,<data_bits>,<stop_bits>**

<port>={COM1,COM2…COM4}

<baud> =9600, 19200, …

<parity> = {N,E,O}

4.4.5 INTERFACE:SCRAMNET:CSR[?]

This command is used to monitor the ScramNet configuration/status registers. For a register descriptions, please refer to *SCRAMNet+ Network PCI Hardware Reference.*

COMMAND: **:Interface:SCRAMNet:CSR? <register>,<value>**

- **<register> [NR1] {CSR0, CSR1, CSR2, … , CSR15}**

QUERY: **:Interface:SCRAMNet:CSR? < register>**

→ <value>

Status Bit Meanings – SCRAMNet+

Bit	Name	Description
00	Transmit FIFO Full	See CSR01 in Systran manual.
01		
02	Transmit FIFO 7/8 Full	See CSR01 in Systran manual.
03	SCRAM_SPURIOUS_INTERRUPT	An incorrect network address was detected in the Demand Sync ID message.
04	Interrupt FIFO Full	See CSR01 in Systran manual.
05	Protocol Violation	See CSR01 in Systran manual.
06	Carrier Detect Fail	See CSR01 in Systran manual.
07	Bad Message	See CSR01 in Systran manual.
08	Receiver Overflow	See CSR01 in Systran manual.
09	Transmitter Retry	See CSR01 in Systran manual.
10	Transmitter Retry Due to Time-out	See CSR01 in Systran manual.
11	SCRAM_INTERRUPT_RETRY	A Demand Sync ID message had its RETRY bit set.
12	SCRAM_UNEXPECTED_INTERRUPT	More than one entry was found in the Interrupt FIFO.
13	SCRAM_BAD_DTYPE	A data item is not of a required type.
14	SCRAM_BAD_SYNCID	The received SyncID contained error bits.
15	SCRAM_DATA_READY	Monitor or Demand Data is Ready

4.4.6 INTERFACE:SCRAMNET:MODE[?]

COMMAND: **:Interface:ScramNet:Mode <mode>**

☞ **<mode>** [enum] {Monitor | Wire | Mechanical | Fiber | Insert}

EXAMPLE: :Interface:Scram:Mode Insert

:Interface:Scram:Mode?

→ Insert

ERRORS:

1. ScramNet Board was not detected.

Notes:

Only "Insert" and "Monitor" modes are enabled. The power up mode is "Monitor". The host program must command "Insert" mode in order to insert the ACT3000 into the SCRAMNet network.

4.4.7 INTERFACE:SCRAMNET:RESET

COMMAND: **:Interface:ScramNet:Reset**

EXAMPLE: :Interface:Scram:Mode Insert
:Interface:Scram:Reset

ERRORS:

1. ScramNet Board was not detected.

Notes:

4.4.8 INTERFACE:VMIC[?]

This command is used to set parameters specific to the VMIC reflective memory interface.

COMMAND: :Interface:Vmic <channel>,<network interrupt node>,<network interrupt>, <interrupt data>

- **<channel>** [enum] {RT | ACL} Specifies the channel to which the remaining parameters apply.
- **<network interrupt node>** [NR1] {0..255} This node will be interrupted on each write for the specified channel.
- **<network interrupt>** [enum] { None | 1 | 2 | 3} Which network interrupt to use. None means no interrupt will be sent.
- **<network interrupt data>** [NR1] {-32768..32767} 33-bit data that will be sent with each network interrupt.

QUERY: :Interface:Vmic:CSR? < channel>

→ <network interrupt node>,<network interrupt>, <interrupt data>

4.4.9 INTERFACE:VMIC:CSR[?]

This command is used to monitor and change the VMIC CSRS configuration registers. Not all registers can be modified; please refer to *VMIPCI-5565 Ultrahigh-Speed Fiber-Optic Reflective Memory with Interrupts.*

COMMAND: **:Interface:VMIC:CSR <register>,<value>**

- **<register>** [NR1] "{BRV | BID | NID | LCSR1 | LISR | LIER | NTD | NTN | NIC |… …ISD1 | SID1 | ISD2 | SID2 | ISD3 | SID3 | INITD | INITN}".
- **<value>** [NR1]

EXAMPLE: :Interface:VMIC:CSR “BRV”,128

QUERY: **:Interface:Vmic:CSR? < register>**

→ <value>

EXAMPLE: :Interface:VMIC:CSR “BRV”

→ 128

4.4.10 INTERFACE:VMIC:RESET

COMMAND: **:Interface:Vmic:Reset**

EXAMPLE: :Interface:Vmic:Reset

ERRORS:

1. VMIC Board was not detected.

Notes:

4.4.11 INTERFACE:REFLECTIVEMEM:SETUP[?]

COMMAND: **:Interface:ReflectiveMem:Setup <dem_blk_addr>,<dem_sync_ID_addr>, <mon_blk_addr>,<mon_sync_ID_addr>,<acl_cmd_blk_addr>,<acl_cmd_sync_ID_addr>,<acl_rsp_blk_addr>,<acl_rsp_sync_ID_addr>**

- **<dem_blk_addr>** [NR1] {0..#H7FFFFC} Demand Block Network Address
- **<dem_sync_ID_addr>** [NR1] {0.. #H7FFFFC} Demand Sync ID Network Address
- **<mon_blk_addr>** [NR1] {0.. #H7FFFFC} Monitor Block Network Address
- **<mon_sync_ID_addr>** [NR1] {0.. #H7FFFFC} Monitor Sync ID Network Address
- **<ACL_cmd_block_addr>** [NR1] {0.. #H7FFFFC} ACL Command Block Network Address
- **<ACL_cmd_sync_ID_addr>** [NR1] {0.. #H7FFFFC} ACL Command Sync ID Network Address
- **<ACL_rsp_block_addr>** [NR1] {0.. #H7FFFFC} ACL Response Block Network Address
- **<ACL_rsp_sync_ID_addr>** [NR1] {0.. #H7FFFFC} ACL Rseponse Sync ID Network Address

EXAMPLE: :Interface:Refl #H4D225C, #H4D235C, #H4D205C, #H4D215C; This example changes the Real-time addresses only.

EXAMPLE: :Interface:Refl #H4D225C, #H4D235C, #H4D205C, #H4D215C, #H3D225C, #H3D235C, #H3D205C, #H3D215C; This example changes both RealTime and Non-real-time ACL addresses.

ERRORS:

1. Reflective Memory Interface was not detected.
2. ScramNet I/O blocks not within a single 4k page.

Notes:

1. A status word :Status:ScramNet is provided to monitor the operation of the ScramNet interface.

See Interface:CSRS for setting the ScramNet configuration registers.

4.5 INTERLOCK

The INTerlock Subsystem commands provide the means to activate the actuation systems for each axis so that the servoed modes of operation can be utilized. This is accomplished in conjunction with the instrumentation, motion system, and facility safety interlocks.

4.5.1 INTERLOCK:CLOSE

The **Interlock:Close** command instructs Acutrol to close the safety interlock string and servo the specified axis. If the **Close** command is not successful at servoing the axis, then a Service Request can be generated. The enabling of the SRQ is a command in the UTILity subsystem.

COMMAND: **:Interlock:Close <axis>**

EXAMPLE: :INT:CLOSe 1
:I:C 1;C 2

Errors:

1. Invalid <axis>. Set the PARAMETER OUT OF RANGE bit.

4.5.2 INTERLOCK:CONFIGURE[?]

This command defines the source of signals used for fail-safe interlock testing, and defines the delays associated with interlock testing and drive enabling.

This feature is currently implemented using configuration parameters: <sel_intlk_states> is hard coded, <fault_delayA> corresponds to parameter x027,<fault_delayB> corresponds to parameter x028, and <drive_enable_delay> corresponds to parameter x029.

COMMAND: **:Interlock:Configure <axis>,<sel_intlk_states>,<enbl_SIL_fault>,<fault_delayA>,<fault_delayB>,…**
…<drive_enable_delay>

- **<axis>** [NR1] = {1|2|…|n}; where [n] is the maximum number of axes.
- **<sel_intlk_states>** [enum] {Local | Multiplex} defines the source of interlock signals that are used for the fail-safe interlock testing. "L" => Local states are sourced from the GPIO pins, the Aux-servo connector, and from internal AIM-FPGA test signals. "M" =>Multiplex states are sourced from the Multiplex Input Register (MIR) whenever a multiplexed I/O expansion module is incorporated.
- **<enbl_SIL_fault>** [enum] {Enable | Disable} Enables/Disables a servo fault condition on one axis to alert other axes by pulling the (open drain) Servo Interlock Link line in the AIM Sync connector.
- **<fault_delayA>** [NR2] {6553.5 seconds, 0.1 second resolution} Defines a generic delay time for the fail-safe axis interlock test.
- **<fault_delayB>** [NR2] {6553.5 seconds, 0.1 second resolution} Defines another generic delay time for the fail-safe axis interlock test.
- **<drive_enable_delay>** [NR2] {6553.5 seconds, 0.1 second resolution} Defines the time delay between issuing an Interlock:Close command and when the Drive Enable(0) is issued. Fault testing is delayed correspondingly. Drive Enable(1) is issued without delay.

QUERY: **:Interlock:Configure? <axis>**

→ <sel_intlk_states>,<fault_delayA>,<fault_delayB>,<drive_enable_delay>

4.5.3 INTERLOCK:CONTROL[?]

This command is not implemented at this time and legacy commands must be used instead.

This is an Acutrol3000 command that consolidates all interlock control features into one command. Commands can be issued to one or all axes. This permits resetting faults and enabling live status on an axis by axis basis.

COMMAND: **:Interlock:Control < axis|ALL>,<state>**

- <**state**> [enum] {Close|Open|Reset} Set interlock state to Close, Open, or Reset

Query: **:Interlock:Control? <axis>**

→ <state>

- <**state**> [character] {Close|Open|Fault} Close, Open, Fault

4.5.4 INTERLOCK:LOGIC[?]

This command is used to customize the safety interlocks for a system/axis.

This feature is currently implemented using parameter commands. Parameter commands require that data for all bits be issued it once. The interlock nomenclature is not implemented at this time. The <enable_mode> argument is implemented using a combination of masks: Intlk_Mask (enable) is Parameter x024, Delay1_Mask(A) is parameter x025, Delay2_Mask(B) is parameter x026, and <sense> is configured using Intlk_Sense which is parameter x023.

COMMAND: :Interlock:Logic <axis>,<intlk_bit>,<"Interlock Name">,<enable_mode>,<sense>

- **<axis>** [NR1] = {1|2|…|n}; where [n] is the maximum number of axes.
- **<intlk_bit>** [NR1] {0..15} Specifies the interlock signal/bit that is to be configured.
- <"Interlock Name"> [string – 26 characters max] Defines the 'user name' for the interlock signal.
- **<enable_mode>** [enum] {Adelay | Bdelay | NoDelay | Disable} Defines if a signal is included in the fail-safe hardware interlock test, and if so, what delay to use. "Adelay" => enabled for test using the Fault_DelayA, "Bdelay" => use Fault_DelayB, "NoDelay" => enabled with no delay, and "Disable" => this interlock signal will not be tested. Note, interlock bits can be monitored in the corresponding status words regardless of their enable configuration.
- **<sense>** [enum] {+ | - } Defines the logic adjustment required to make an interlock signal have a FAULT = TRUE logic. "+" => no inversion, "–" => invert signal before testing.

Fail Safe Interlock Configuration				
			SEL_INTLK_STATES	
Bit#	Intlk_Status2	Interlock_Name (User Definable)	Local	Multiplexed
0	INTLK_STATUS0	Intlk1	GPIO0	MIR0
1	INTLK_STATUS1	Intlk2	GPIO1	MIR1
2	INTLK_STATUS2	Intlk3	GPIO2	MIR2
3	INTLK_STATUS3	PGOUT3	GPIO3	MIR3
4	INTLK_STATUS4	GPOUT4/Global3	GPIO4	MIR4
5	INTLK_STATUS5	GPOUT5/Global4/Intlk4	GPIO5	MIR5
6	INTLK_STATUS6	PA_Ack	(Aux Servo) PA_ACK0	MIR8
7	INTLK_STATUS7	ASIntlk1/Global1	(Aux Servo) INTLK1	MIR9
8	INTLK_STATUS8	ASIntlk2/Global2	(Aux Servo) INTLK2	MIR10
9	INTLK_STATUS9	MstrIntlk&FPDsbl	(Aux Servo) MASTER_INTLK	MIR11
10	INTLK_STATUS10	AIM_FAULT - DMA_SYNC_ERROR (DMA Cycle Fault)		
11	INTLK_STATUS11	AIM_FAULT - SW_WD_FAULT (Software Watchdog Fault)		
12	INTLK_STATUS12	AIM_FAULT - CLK_WDOG (10 KHz Bus Clock Watchdog)		
13	INTLK_STATUS13	AIM_FAULT - SDL_FAIL (Serial Data Link Fault)		
14	INTLK_STATUS14	AIM_FAULT - SIL (Servo Interlock Link)		
15	INTLK_STATUS15	AIM_FAULT - 10K_LOCK_FAULT (10KHz Phase Lock Error)		

4.5.5 INTERLOCK:OPEN

The OPEN command tells Acutrol that the specified axis is to be shut down in an orderly fashion.

COMMAND: :Interlock:Open <axis>

EXAMPLE: :INT:OPEN 1
:I:O 1;O 2;O 3

Errors:

1. Invalid <axis>. Set the PARAMETER OUT OF RANGE bit.

4.5.6 INTERLOCK:RESET

When an axis interlock string is broken the interlock status is frozen so that the offending event can be detected and isolated. This is necessary as many faults may be indicated as an axis shuts down. After the frozen status information has been read the :INTerlock:RESet command will unfreeze the status on all axes that are faulted so that current status may be monitored.

This single command will clear the frozen interlock state on all axes in the system so that live status can be read, and axes can be servoed. This command will have no effect on an axis that is already servoed.

COMMAND: :Interlock:Reset

EXAMPLE: :INT:RES

:INT:RES;CLOS 1

:D:P 2,0.0;:I:R;:M:P 2;:I C 2

(Set position on axis2 to 0.0; Reset interlock faults; Change axis 2 to Position mode; Close axis 2 interlocks and servo).

4.6 LIMIT[?]

The Limit commands define the extreme values of the profiled motion states that may be produced by the Command Generator (Limiter). The purpose of imposing limits on the motion command states is to ensure that saturation is prevented at all points in the servo system. A saturated system cannot be controlled and this condition is potentially hazardous to equipment and personnel.

All limits in this command group are entered using the user defined engineering units defined in Configure:Scaling section □.

Some limits represent maximum or minimum values, while others represent absolute magnitudes. Each limit command has an associated Command Profiler mode. These are single characters indicate the desired mode:

A = Abort Command Processor Mode (used internally by the ACP).
O = Off (Zero Rate) Command Processor Mode.
P = Position Command Processor Mode.
R = Rate Command Processor Mode.
S = Synthesis Command Processor Mode.
T = Track Command Processor Mode.

COMMAND: **:Limit: <axis|ALL>,<mode>,<rate_lim>,<acc_lim>,<jerk_lim>**

- **<axis>** [NR1] Specifies the axis to be modified.
- **<mode>** [enum] {Position | Rate | Arate | Synthesis | Track | Abort | Off} Defines the profiler mode for the specified limits that follow.
- **<rate_lim>** [NRf] Mode specific limit on the velocity motion state.
- **<acc_lim>** [NRf] Mode specific limit on the acceleration motion state.
- **<jerk_lim>** [NRf] Mode specific limit on the jerk motion state.

4.6.1 LIMIT:ABSOLUTE[?]

Absolute limits define the absolute maximum value that can be set for the working limits of the Motion Command Processor. They are in fact limits on the limits and are generally set at the factory and cannot be changed by a low security interface/user. In the GUI, these limits are referred to as the Factory Limits.

COMMAND: **:Limit:Absolute <axis|ALL>,<rate_lim>,<acc_lim>,<jerk_lim>[,<pos_lo_lim>,<pos_hi_lim>, <Vtrip>,<tracking_error_trip>**

- <**axis**> [NR1] Specifies the axis to be modified.
- <**rate_lim**> [NR2] Absolute maximum value that the Rate Limit can be set.
- <**acc_lim**> [NR2] Absolute maximum value that the Acceleration Limit can be set.
- <**jerk_lim**> [NR2] Absolute maximum value that the Jerk Limit can be set.
- <**pos_lo_lim**> [NR2] Sets the minimum low limit for the position motion state. The low limit does not need to be negative, but must be less than the <pos_hi_lim>.
- <**pos_hi_lim**> [NR2] Sets the maximum high limit for the position motion state. The high limit does not need to be positive, but must be greater than the <pos_lo_lim>.
- <**Vtrip**> [NR2] Absolute maximum value that the Velocity Trip Limit may be set.
- <**tracking_error_trip**> [NR2] Absolute maximum value that the Tracking Error Trip limit may be set.

4.6.2 LIMIT:ACCELERATION[?]

The ACCeleration Limit command specifies the maximum absolute value that the acceleration command state variable can assume. The Command Processor uses this and other limits to constrain the dynamic range of motion command states while maintaining their integral relationship. An Acceleration limit of zero (0.0) should never be commanded because all changes to motion command states would be inhibited.

This ACL command is provided for Acutrol Act2000 compatibility.

COMMAND: :Limit:Acceleration <mode>,<axis>,<acceleration limit>

EXAMPLE: :LIM:ACC P,1,1500.0

:LIM:H P,4,180.0;L P,4,-180.0;R P,4,100.0;A P,4,2000.0

QUERY: :Limit:Acceleration? <mode>,<axis>

→ <acceleration limit>

EXAMPLE: :LIM:ACC? P,1

→ 1500.0

:LIM:H P,4,180.0;L P,4,-180.0;R P,4,100.0;A P,4,2000.0

→ 180.0;-180.0;100.0;2000.0

Errors:

1. Invalid <mode>,<axis>. Set the PARAMETER OUT OF RANGE bit.

4.6.3 LIMIT:BEMF[?]

This command provides two or four quadrant torque-speed curve limiting. The operating envelope is defined by the user specified acceleration and rate limits for the current Command Processor mode and by a torque speed line that is defined by rate and acceleration intercepts. Refer to TM-9226. This command is Demand scaled, i.e. the scaling and engineering units associated with the command arguments are consistant with the scaling and units of the rate and acceleration motion damands in the system.

COMMAND: :Limit:Bemf <axis>,<rate_x>,<accel_x>,<mode>

- <axis> **[NR1] = {1|2|…|n}; where [n] is the maximum number of axes.**
- <rate_x> **[NRf]/[NR2] Defines the theoretical peak velocity at zero acceleration.**
- <accel_x> **[NRf]/[NR2] Defines the theoretical peak acceleration at zero velocity.**
- <mode> **[NRf]/[NR1] {0|2|4} Defines the limiting mode, where 0 = Disabled, 2 = two quadrant, and 4 = four quadrant operation.**

EXAMPLE: :Limit:BEMF 2,1200,2000,2

Query: :Lmit:Bemf? <axis>

→ <rate_x>,<accel_x>,<mode>

4.6.4 LIMIT:GLOBAL[?]

COMMAND: **:Limit:Global <axis:ALL>,<pos_lo_lim>,<pos_hi_lim>,<Vtrip>,…**
…<tracking_error_trip>[,<track_var>]

- <**axis**> [NR1] Specifies the axis to be modified.
- <**pos_lo_lim**> [NR2] User specified low limit on the position motion state. The low limit does not need to be negative, but must be less than the <pos_hi_lim>.
- <**pos_hi_lim**> [NR2] User specified upper limit on the position motion state. The high limit does not need to be positive, but must be greater than the <pos_lo_lim>.
- <**Vtrip**> [NR2] User specified limit on the Velocity Trip. If system rates exceed this maximum limit, then a fault will be generated and the servoed axis will be Aborted.
- <**tracking_error_trip**> [NR2] User specified limit on the position tracking error of the servo loop. Exceeding this maximum limit will generate a fault and the servoed axis will be Aborted.
- <**track_var**> [NR2] This reference specifies which variable the Tracking Error Test is applied.

4.6.5 LIMIT:GLOBAL:METHOD[?]

COMMAND: :Limit:Global:Method <axis:ALL>, <limiterMethod>

- **<limiterMethod>** [Null | Accel1 | Jerk1]] This specifies which limiter algorithm is applied.

4.6.6 LIMIT:HIGHPOSITION[?]

The High POSition Limit command specifies the maximum value that the position command state variable can assume. The Command Processor uses this and other limits to constrain the dynamic range of motion command states while maintaining their integral relationship.

Note: It is illegal to have HPOS numerically greater than LPOS . This limit is used only if the ACP is set for **LIMITED** motion. See Section 4.2.1

This ACL command is provided for Acutrol Act2000 compatibility.

COMMAND: :Limit:Highposition <mode>,<axis>,<high position limit>

EXAMPLE: :LIM:HPOS P,2,179.0

:L:H R,3,45.0;H R,1,90.0

:L:H 1,-180

A High POSition Limit query returns the current position limit.

QUERY: :Limit:Highposition? <axis>

→ <high position limit>

EXAMPLE: :LIM:HPOS? P,3

→ 179.90000 (Format ASCII data)

:L:H? 2

→ #14<4 bytes high position limit data> (Format Binary data)

Errors:

1. Invalid <mode>,<axis>. Set the PARAMETER OUT OF RANGE bit.

4.6.7 LIMIT:LOWPOSITION[?]

The Low POSition Limit command specifies the minimum value that the position command state variable can assume. The Command Processor uses this and other limits to constrain the dynamic range of motion command states while maintaining their integral relationship.

Note: It is illegal to have LPOS numerically less than HPOS .

This limit is used only if the ACP is set for **LIMITED** motion. See Section 4.2.1

This ACL command is provided for Acutrol Act2000 compatibility.

COMMAND: :Limit:Lowposition <mode>,<axis>,<low position limit>

EXAMPLE: :LIM:LPOS S,2,-179.0

:L:L T,3,-45.0;H T,3,45.0

:L:L P,1,50.0;H P,1,100.0

A Low POSition Limit query returns the current position limit.

QUERY: :Limit:Lowposition? <axis>

→ <low position limit>

EXAMPLE: :LIM:LPOS? P,3

→ -45.00000

:L:L? 2

→ #14<4 bytes low position limit data>

Errors:

1. Invalid <mode>,<axis>. Set the PARAMETER OUT OF RANGE bit.

4.6.8 LIMIT:RATE[?]

The RATE Limit command specifies the maximum absolute value that the rate command state variable can assume. The Command Processor uses this and other limits to constrain the dynamic range of motion command states while maintaining their integral relationship.

This ACL command is provided for Acutrol Act2000 compatibility.

COMMAND: :Limit:Rate <mode>,<axis>,<rate limit>

EXAMPLE: :LIM:RATE P,1,120.0

:L:R R,3,100.0;R R,2,10.0

:L:R P,5,#14<4 bytes binary rate limit>

Note the Rate limit has no significance for the Abort and Off (Zero Rate) Modes. In these modes the rate limit is forced to zero and the axis decelerates to zero velocity at the mode's acceleration limit.

A RATE Limit query returns the current rate limit for the selected mode.

QUERY: :Limit:Rate? <mode>,<axis>

→ <rate limit>

EXAMPLE: :LIM:RATE? P,3

→ 50.0000

:L:R? T,2

→ #14<4 bytes rate limit data>

Errors:

1. Invalid <mode>,<axis>. Set the PARAMETER OUT OF RANGE bit.

4.6.9 LIMIT:VELOCITYTRIP[?]

The Velocity TRip command specifies the set point for detecting over-rate conditions. This safety feature looks at the (estimated) rate feedback, and operates independently of the Command Processor.

Upon detection of a velocity trip, the axis will shut down on the assumption that a run-away condition exists.

This ACL command is provided for Acutrol Act2000 compatibility.

COMMAND: **:Limit:Vtrip <mode>,<axis>,<velocity trip>**

EXAMPLE: :LIM:VTR R,2,100.0

:L:V P,2,110.0;:DEM:RATE 2,100.0

:L:V P,3,1000.0

A Velocity TRip query returns the current rate trip for the selected axis.

QUERY: **:Limit:Vtrip? <mode>,<axis>**

→ <rate trip>

EXAMPLE: :LIM:VTRip? P,3

→ 150.0000

:L:V? 2

→ #14<4 bytes rate trip data>

Errors:

1. Invalid <mode>,<axis>. Set the PARAMETER OUT OF RANGE bit.

4.7 MODE[?]

The MODE Subsystem commands listed below describe the servo control modes that are available in the Acutrol3000 Motion Controller. Only one command is required to command and query the mode of an axis. The sub-sections define Acutrol Act 2000 compliant commands that function as well. All axes can be changes with a single command, but each axis must be queried individually.

COMMAND: **: Mode <axis|ALL>,<mode>**

- **<axis> [NR1]** is the axis number
- **<mode>** [enum]{Position | Rate | Arate | Synthesis | Track | Abort | Off } is one of the following:
  - Position
  - Rate
  - ARate
  - Synthesis
  - Track
  - Off

A MODE query returns the current mode of the selected axis.

QUERY: **:Mode? <axis>**

→ <mode>

EXAMPLE: :MODE? 2

→ P

:M? 1;:M? 2;:M? 3

→ P;R;P

Errors:

1. Invalid <axis>. Set the PARAMETER OUT OF RANGE bit.

4.7.1 MODE:ARATE

Absolute RATE servo mode provides closed loop control of an axis such that stable steady state velocity is maintained. Changing the rate demand while in Rate Mode will profile the motion states to the desired rate with constrained acceleration and jerk.

Absolute rate allows an overshoot of the rate setpoint to allow the axis to achieve a position sychronization that results from no acceleration or jerk limit.

COMMAND: :Mode:Arate <axis>

EXAMPLE: :Mode:Arate 3

Errors:

1. Invalid <axis>. Set the PARAMETER OUT OF RANGE bit.

4.7.2 MODE:HOME

Change this to a non-mode command.

Note: Change definition to "Initiate Homing Sequence defined with the :CONFIG:HOME command".

COMMAND: **:Mode:Home <axis>**

EXAMPLE: :Mode:Home 3

:M:H 1;H 2;H 3

Errors:

1. Invalid <axis>. Set the PARAMETER OUT OF RANGE bit.

4.7.3 MODE:OFF

Off servo mode provides closed loop control of an axis such that a stable steady state of zero velocity is maintained. No demands affect Off Mode. An axis may be changed to Off Mode while in motion to bring the axis to a smooth controlled stop. This mode is equivalent to Rate Mode with a demand of 0.

COMMAND: **:Mode:Off <axis>**

EXAMPLE: :MODE:OFF 1

:M:O 2;O 3

Errors:

1. Invalid <axis>. Set the PARAMETER OUT OF RANGE bit.

4.7.4 MODE:POSITION

Position servo mode provides closed loop control of an axis such that stable steady state positioning is maintained. Changing the position demand while in Position Mode will result in profiled motion commands to move to the desired position with constrained velocity, acceleration, and jerk.

Systems can be either Continuous or Limited motion; all limited motion systems observe the maximum and minimum position limits as specified by the LIMit subsystem commands. Linear motion systems are by definition limited motion; rotary systems can be either. Position commands for continuous rotational axes result in shortest distance or "least path" profiles. Refer to the Operator Interface Manual for configuring Continuous and Limited systems.

COMMAND: :Mode:Position <axis>

EXAMPLE: :MODE:POS 3

:M:P 2

:M:P 1;P 2

Errors:

1. Invalid <axis>. Set the PARAMETER OUT OF RANGE bit.

4.7.5 MODE:RATE

Relative RATE servo mode provides closed loop control of an axis such that stable steady state velocity is maintained. Changing the rate demand while in Rate Mode will profile the motion states to the desired rate with constrained acceleration and jerk.

The axis will achieve the demanded rate as quickly as possible, without any overshoot, given the physical constraints of the system and programmed limits.

COMMAND: **:Mode:Rate <axis>**

EXAMPLE: :MODE:RATE 3

Errors:

1. Invalid <axis>. Set the PARAMETER OUT OF RANGE bit.

4.7.6 MODE:SYNTHESIS

This mode synthesizes sinusoidal position, rate, acceleration, and jerk motion states. The amplitude, frequency, and phase of the sinusoidal motion are defined with :DEMand:OSCillator. These demands should be loaded before starting SYNThesis mode.

It is desirable to enter SYNThesis Mode from POSition Mode as the position demand establishes the center of the sinusoidal motion. Switching to another mode such as POSition or OFF Mode terminates SYNThesis Mode.

Multiple axes can be relatively scaled and phased to produce complex motion. To start multiple axes synchronously the mode commands must be in the same device dependent message.

{Future: It would be nice to have a synthesis mode that used constant amplitude rate or acceleration sinewaves instead of position sinewaves, this can be added later}

COMMAND: :Mode:Synthesis <axis>

EXAMPLE: :MODE:SYNT 1

:M:S 2;S 4;R 3;P 1

Errors:

1. Invalid <axis>. Set the PARAMETER OUT OF RANGE bit.

4.7.7 MODE:TRACK

TRACk mode is used to follow real-time motion states generated by the Host Computer. Demands are expected in a vector form (see :DEMand:LONGvector, VECT, and SVEC) and received on a periodic time basis. This command is also used in conjunction with the real-time parallel and shared memory interfaces.

More details on Track Mode and the related issues of real-time interface operation/configuration are discussed in TM-9374 Acutrol3000 Real-time Data Communication.

The optional <interval> argument defines the number of milliseconds between demands. If demand data is not received as expected then the axis is switched to OFF Mode. If the <interval> argument is omitted, the interval defaults to 1 ms. The <interval> argument has a range of 0 to 65,535 ms and is specified in <NRf> format. The value of 0 disables the interval timeout checking.

TRACk mode may end in a track mode time-out as a result of exceeding the timeout <interval>. The CIC can attempt to switch the mode (to OFF, POSition, RATE, or SYNThesis), but with a 1 ms interval the mode switch may not commence within the time-out period. Using a second remote interface along with the IEEE-488 allows the download of motion demands to continue while the servo mode is changing, preventing a TRACk mode time-out.

The test for a track mode time-out does not begin until the first demand is received after switching to TRACk Mode.

COMMAND: :Mode:Track <axis>,<interval>

EXAMPLE: :MODE:TRAC 1

:M:T 2;T 3

:M:T 1,5;T 2,10

4.8 PLAYBACK

The **Playback** Subsystem provides commands to playback realtime data with the Acutrol3000.

4.8.1 PLAYBACK:ACTIVATE[?]

The Activate command activates the Playback processes. The Activate command has a single parameter that may take on any of six values.

COMMAND: **:Playback:Activate {ARM | NOW | WAIT | STOP | ABORT | OFF}**

ARM – Commence playback upon satisfaction of the triggering criteria specified using the trigger subcommand.

NOW – Commence playback immediately.

WAIT – Commence playback upon receipt of the *TRG common command or, if using the GPIB interface, the GET multi-line command.

STOP – End playback at the completion of the current data set.

ABORT – End playback immediately.

RESET – End a previously started playback and reset the index to 0.

QUERY: **:Playback:Activate?**

4.8.2 PLAYBACK:CONFIGURE[?]

The Configure command is used to specify the basis for data playback.

COMMAND: **:Playback:Configure <iterations>**

- <iterations> **– The number of time to playback the buffer or** CONT

4.8.3 PLAYBACK: DATA [?]

This allows playback data to be sent to the Acutrol3000.

COMMAND: **:Playback:Data <axis>,<data>**

- ☞ <data> is definite length binary data. The data is sent as IEEE754.1 Floating Point, Double Precision. The first 1/3 of the file is position demand data. The second 1/3 is rate demand data, and the last 1/3 is acceleration demand data.

4.8.4 PLAYBACK:LIST?

Returns a list of all available playback files stored on the hard drive.

COMMAND: **:Playback:List ? [< version >]**

4.8.5 PLAYBACK:RESTORE[?]

This allows playback data to be restored from the local hard drive.

COMMAND: **:Playback:Restore [< version >]**

- **<version >** – is a three digit numeric reference for the saved data. 000 is the default.

4.8.6 PLAYBACK:SAVE[?]

This allows playback data previously sent by the :Playback:Data command to be saved to the local hard drive.

COMMAND: **:Playback:Save [,< version >[,<comment>]]**

- **<version >** – is a three digit numeric reference for the saved data. 000 is the default.
- <comment> is a ASCII string description of the LUT contents and is limited to 72 characters.

4.8.7 PLAYBACK:TRIGGER[?]

The TRIGger feature of the Playback command is used to specify the basis of the start event, which initiates the playback process.

COMMAND: **:Playback:TRIGger <variable>[,<level>[,<sign>]]**

- < **variable** > – The trigger variable.
- < **level** > – The trigger threshold value. <level> is in the same units as the variable associated with the trigger channel. This is combined with the <sign> parameter to determine when triggering will occur. Default is 0.
- <**sign**> – The trigger sense, '+' or '-'. Used to specify the direction that the trigger channel's value should be going when triggering occurs. Default is +.

4.9 QUERY

4.9.1 QUERY:ERROR?

COMMAND: **:Query:Error? <error code>**

→<Description of error (text string)>

4.9.2 QUERY:SYSTEM?

COMMAND: **:Query:System?**

→ <Number of Axes>,<First axis number>,…<Last axis number>

4.9.3 QUERY:VARNUM?

COMMAND: **:Query:TotalVariables?**

→<Total number of variables>

4.10 READ

The READ Subsystem commands are used to access system variables in Acutrol. The commonly accessed motion state variables are provided with dedicated commands to make reading them convenient.

The <data> response to the READ commands will be sent as 32-bit signed integers or as decimal numbers (see section on data formats). The format for response data depends on the format specified by the :Config:Variable command.

4.10.1 READ:ACCELERATION?

Selects the axis acceleration variable and responds with the appropriate acceleration feedback data.

COMMAND: **:Read:Acceleration ?<axis>**

→ <axis acceleration data>

EXAMPLE: :READ:ACC 1

→ <axis 1 acceleration data>

:R:A 2;R 2;P 2

→ <axis 2 acceleration data>;<axis 2 rate data>;<axis 2 position data>

:READ:ACC 2;RATE 2;POS 2

→ #14<4 bytes axis 2 acceleration data>;#14<4 bytes axis 2 rate data>;…

…#14<4 bytes axis 2 position data>

Errors:

1. Invalid <axis>. Set the PARAMETER OUT OF RANGE bit.

Note: For ACT2000 compatibility, the ? is not required.

4.10.2 READ:LOGDATA?

(Possible future feature; currently logged data is retreived directly from the real-time computer using an FTP client).

This command is used to read the data logged by the ACQuire Subsystem. The <channel> argument specifies the channel of logged data. Acutrol responds with a serial stream of data starting with the first logged point <first data> and ending with the last point <last data>.

The number of data points that may be read at one time is limited by the size of the Acutrol output buffer. The interface computer (CIC) continues to issue :READ:LOGData commands until all data is collected. The logged data will be broken up across multiple buffers between data points and each logged variable is read independently. Data are not destroyed when read and may be re-read until the log channel is Freed, or a new set of data is logged. Data can not be read while data is being logged.

Note: Data is always returned in the format specified by the :CONFIG:VAR command for the corresponding variable for <channel>. In the response, num_pts indicates the actual amount of data points returned in the response and will always be <= the number of points requested. Issuing a command without the <index> and <num_pts> fields will return the next <num_pts> of data. If <num_pts> is never specified, it will default to the maximum available for the output buffer.

COMMAND: **:Read:Logdata? <channel>[,<index>,<num_pts>]**

→<index>,<num_pts>,<first data>,…,<last data>

EXAMPLE: :READ:LOGD? 3, 0,4096

→ 0,2048,100.0012,...,105.0341 (Decimal format)

Errors:

2. Invalid <channel>,<index>,<num_pts>. Set the PARAMETER OUT OF RANGE bit.

Note: The maximum value for Num_pts may need to be interface specific and also depends on the output data type.

4.10.3 READ:VARIABLE?

The Variable subcommand is a read (only) command that is used to monitor the current value of Acutrol3000 System Variables. These are the same "user" variables that are assigned a variable number, and are referenced in the system allowing flexible configurations, state monitoring, output to analog signals, etc.

Note: :R:V defaults to :READ:VECTOR to maintain compatibility.

Command: :Read:Variable? <variable>

→ <value>

- **<variable>** [NR1] System Variable number; where the axis "sub-system" supplying the variable is MOD_{1000}(<variable>), and the specific variable is REM_{1000}(<variable>). You can specify up to 8 variables to read at a single time.

Note: Move the definition of system variable to a "definition" section.

- **<value>** [NR2] response data corresponding to the value of the specified system variable at the time that the data was processed. (All response data from a multi-command command, is collected from the same ACP frame. This ensures that the data is coherent; also, one of the commands can read a time clock or frame count variable so that the data is time tagged and/or sequenced.

Command: :Read:Variable? <variable>

→ <value>

EXAMPLE: : R:V 3100 (read ACP variable x100 on axis3)

→ 123.4 (ASCII Format)

:R:V? 1021;V? 2083

→ #14<4 byte value>;#14<4 byte value> (Binary Format)

Errors:

1. Invalid <variable>. Set the PARAMETER OUT OF RANGE bit.

4.10.4 READ:POSITION?

Selects the axis position feedback variable and responds with the appropriate data. The <axis> argument is an integer (1 to 6) in ASCII format.

COMMAND: :Read:Position? <axis|ALL>

→ <axis position data>

EXAMPLE: :READ:POS? 2

→ <axis 2 position data>

:READ:POS? 1;POS 2;POS 3

→ <axis 1 position data>;<axis 2 position data>;<axis 3 position data>

:R:P? 1;P? 2;P? 3 (binary response)

→ #14<4 bytes axis 1 position data>;#14<4 bytes axis 2 position data>;…

…#14<4 bytes axis 3 position data>

:R:P? 1;P? 2;P? 3 (decimal response)

→ 10.0001;44.1250;0.0001

Errors:

1. Invalid <axis>. Set the PARAMETER OUT OF RANGE bit.

Note: For ACT2000 compatibility, the ? is not required.

4.10.5 READ:RATE?

Selects the axis rate variable and responds with the appropriate rate feedback data.

COMMAND: **:Read:Rate? <axis|ALL>**

→ <axis rate data>

EXAMPLE: :READ:RATE? 1

→ <axis 1 rate data>

:R:R? 2;P? 1 (binary response)

→ #14<4 bytes axis 2 rate data>;#14<4 bytes axis 1 position data>

:READ:RATE? 2;POS? 1 (decimal response)

→ <axis 2 rate data>;<axis 1 position data>

Errors:

1. Invalid <axis>. Set the PARAMETER OUT OF RANGE bit.

Note: For ACT2000 compatibility, the ? is not required.

4.10.6 READ:SHORTVECTOR?

The Short VECtor command tells Acutrol to report the motion states (position and rate) for the specified <axis> (1 to 6).

COMMAND: :Read:Svector? <axis|ALL>

→ [#18]<4 bytes axis position data><4 bytes axis rate data>

EXAMPLE: :READ:SVEC? 6 (binary response)

→ #18<4 bytes axis 6 position data><4 bytes axis 6 rate data>

:R:S? 2 (decimal response)

→ <axis 2 position data>;<axis 2 rate data>

Errors:

1. Invalid <axis>. Set the PARAMETER OUT OF RANGE bit.

Note: For ACT2000 compatibility, the ? is not required.

4.10.7 READ:VECTOR?

This command tells Acutrol to report the motion states (position, rate, and acceleration) for the specified axis.

COMMAND: :Read:Vector? <axis|ALL>

→ [#212]<4 bytes axis position data><4 bytes axis rate data>…

…<4 bytes axis acceleration data>

EXAMPLE: :READ:VECT? 2 (binary response)

→ #212<4 bytes axis 2 position data><4 bytes axis 2 rate data>…

…<4 bytes axis 2 acceleration data>

:R:V? 3 (decimal response)

→ <axis 3 position data><axis 3 rate data><axis 3 acceleration data>

Errors:

1. Invalid <axis>. Set the PARAMETER OUT OF RANGE bit.

Note: For ACT2000 compatibility, the ? is not required.

4.10.8 READ:WORLDCOORDS

This command tells Acutrol to report the state of the motion system in world coordinates. A GCT (Global Coordinate Transform) module must be loaded in Acutrol for this command to function. If a GCT module is not loaded, the result of this command is undefined.

The number of parameters returned is equal to the number of degrees of freedom and is system dependent. See the system's GCT documentation for details.

COMMAND: **:Read:WorldCoords**

→ [#nNN]<GCT parameter 1>[;]<GCT parameter 2>[;…]

EXAMPLE: :READ:WOR

→ 45.00000,90.00000,-90.00000 Roll, pitch, yaw

:R:W

→ #224<24 bytes GCT feedback data> X, Y, Z, roll, pitch, yaw

{Shelf this for now}

4.11 STATUS – ACUTROL3000 COMMANDS

The STATus Subsystem commands are used to capture the operational state of the Acutrol3000 instrumentation by reporting the status of the various subsystems. Each subsystem may have several status words that are individually accessed.

The ACL commands summarized in this section are used to configure the basis for reporting errors and changes of state. The commands also provide a means to query the status of the various Acutrol subsystems using a summary status tree structure.

<Register ID> is a mnemonic reference to the various registers that can be the focus of the Status ACL commands in this section.

The following <register ID> mnemonics are defined based on IEEE 488.2:

STB	Standard Status Byte Register
ESE	Standard Event Status Enable Register
ESR	Standard Event Status Register
ESF	Standard Event Status Transition Filter Register
SRE	Service Request Enable Register

The ACUTROL 3000 Status Registers are described below. The table defines the <register ID> mnemonics and identifies the underlying Status Data Structure. Refer to Appendix B for a description of the individual bit status for each condition register.

Mnemonic	Register Description	Summary Bit/Register
SI	System Interlocks Summary Register	Bit0 / STB
ACP	ACP Processes Summary Register	Bit1 / STB
SUP	Supervisor Processes Summary Register	Bit2 / STB
RI	Remote Interfaces Summary Register	Bit3 / STB
OQ	Output Message Queue (MAV)	Bit4 / STB
SMQ	Status Message Queue	Bit7 / STB
A1	Axis1 ACP Summary Register	Bit1 / ACP
A2	Axis2 ACP Summary Register	Bit2 / ACP
A3	Axis3 ACP Summary Register	Bit3 / ACP
A(n)	Axis(n) ACP Summary Register	Bit(n) / ACP
A(n)STATE	Static Operating State	Bit0 / A(n)
A(n)DYN	Dynamic Operating State	Bit1 / A(n)
A(n)ERR	Procedure Errors	Bit2 / A(n)
A(n)BYTE	Byte Test	Bit3 / A(n)
A(n)DIAG	Diagnostic Status	Bit4 / A(n)
A(n)PGOUT	Programmable Output Register	Bit5 / A(n)
A(n)ID_REG	AIM ID register “A1”	Bit6 / A(n)

A(n)LOCAL	AIM Local Status4	Bit7 / A(n)
A(n)SWITCH	AIM Hardware Switch Register	Bit8 / A(n)
G1INTLK	Global Interlock Status (Combined)	Bit0 / SI
G2INTLK	Global Interlock Status (Combined)	Bit1 / SI
A1INTLK	Axis1 Hardware Interlock/Fault Status	Bit2 / SI
A1SWFLT	Axis1 Software Detected Fault	Bit3 / SI
A2INTLK	Axis2 Hardware Interlock/Fault Status	Bit4 / SI
A2SWFLT	Axis2 Software Detected Fault	Bit5 / SI
A3INTLK	Axis3 Hardware Interlock/Fault Status	Bit6 / SI
A3SWFLT	Axis3 Software Detected Fault	Bit7 / SI
…		
A6INTLK	Axis6 Hardware Interlock/Fault Status	Bit12 / SI
A6SWFLT	Axis6 Software Detected Fault	Bit13 / SI
SUP1	Supervisor Operational Status	Bit0 / SUP
SUP2	Supervisor Inter-Axis	Bit1 / SUP
IEEE488		Bit0 / RI
PARALLEL		Bit1 / RI
SCRAMNET		Bit2 / RI
BIT3		Bit3 / RI
TCPIP		Bit4 / RI
QERROR	Error Messages	Bit0 / SMQ
QFAULT	Servo Interlock Event Log	Bit1 / SMQ
QCMMD	Motion Command Log	Bit0 / SMQ
QDATA	Data Log Queue	Bit1 / SMQ
QALL	All Command and Error Messages	Bit2 / SMQ

4.11.1 STATUS:CSR?

A Condition Status Register (CSR) contains discrete bits (True/False) that represent the "real-time" condition of their associated function. A CSR may have 1 to 16 bits; unused bits are returned as zero. There are no ACL or IEEE-488 commands that have a direct effect on the state of the bits in the CSR. Only the underlying state of the Acutrol operation can set or clear the condition bits.

In most cases condition bits represent Acutrol states; however, a condition bit can represent a summary bit from an overlaying secondary status register.

Status:CSR? is a query only command.

The Condition Status Registers are defined in Appendix B.

COMMAND: **:Status:CSR? <register ID>[,<bit>]**

→ [<CSR register status> | <CSR register bit status>]

- **<register ID>** [enum]{ SI | ACP | SUP | RI | ESR | EQ} mnemonic that references a specific CSR. Refer to the register summary in the introduction of this section.
- **<bit>** [char] {1 | 0} When specified, this argument defines a single bit for which status is reported.

4.11.2 STATUS:ESE[?]

The Event (Status) Enable Register is described in §11.4.2.3, pp. 155 of IEEE Std 488.2, and selects which event bits in the corresponding Event Register are able to cause a summary message bit to be set; and consequently, enable the generation of a service request. This command permits setting the states of the ESE register as well as querying the current state.

COMMAND: :Status:ESE <register Id> [<enable8>| [<enable1>,<bit>]]

- <**register Id**> [enum]{ SI | ACP | SUP | RI | ESR | EQ } mnemonic that references a specific CSR. Refer to the register summary in the introduction of this section.
- <**enable8**> [NRf]/[NR1] {0..255} is a byte that defines the states of the ESE register.
- <**enable1**> [NRf]/[NR1] {0 | 1} is a boolean that defines the enable state of the corresponding <bit> in the ESE register.
- <**bit**> [NRf] {0..7} Specifies the ESE bit that is configured by <enable1>.

EXAMPLE: :Status:ESE 5 (Bit 0 and bit 2 are enabled; all other bits are set to disabled).

:STAT:ese 5,0 (Disable bit 5; all other bits are unchanged).

QUERY: :Stat:ESE? <register Id> [<bit>]

→ [<enable1>|<enable8>]

If the argument <bit> is included in the command then a boolean <enable1> will be returned. Otherwise, the full state of the Event Service Enable Register will be returned as the byte <enable8>.

EXAMPLE: :Stat:Ese?

→ 9 (Bit 0 and bit 3 are enabled; all other bits are disabled).

:Status:ESE? 6

→ 1 (Bit 6 of ESE register is set)

4.11.3 STATUS:ESR?

An Event Status Register (ESR) is one of many Acutrol specific registers that contain bit status of the changing condition of the control system.

When the state of a bit in a condition register changes, the corresponding event register bit is updated according to the defined transition filter associated with that bit. This is the only way in which an event register bit can be set.

The event register bits are latching, i.e., once they are set to **TRUE** they cannot be directly cleared. A change in the associated condition register bit does not cause the corresponding event register bit to revert to **FALSE**.

Reading the bits of an event register (or the full register) is the only way to clear set bits.

Query: :Status:ESR? <register ID>[,<bit>]

- **<register ID>** [enum] { SI | ACP | SUP | RI | ESR | EQ } is a mnemonic that identifies a specific ESR.
- **<bit>** [NR1] {0..15} This optional argument is used to reference a specific event bit.

EXAMPLE: :Status:ESR? A1Dyn,2

→0 Bit2 = (+)Acceleration Limit and has not been breached on Axis1.

Note: The Standard Event Status Register is normaly queried using the *ESR? Common command, but can also be read using the :Status:ESR? ESR ACL command.

4.11.4 STATUS:MSG?

This ACL command is used to read a complete message form one of the Output or Error Message queues. In general, a message queue is read as a result of a summary bit being set in a Status Register during normal Acutrol activity. The status bit remains true as long as the queue is not empty. If this command is sent and the queue is empty, then a null string will be returned, and no error will be reported.

Status:MSG? is a query only command.

Query: **:Status:MSG? <qid>[,<Num_Msg>|,ALL]**

- **<qid>** [string] mnemonic that references a specific queue. Refer to the mnemonic summary in the introduction of this section.
- **<Num_Msg>** NR1 [pos integer | All] specified how many messages are to be returned. If the number of messages in the queue is less than the number requested, then the response is shorter than expected; still there is no error message reported.

4.11.5 STATUS:MSGCLR

Normally reading all of the available messages clears a message queue. This command is used to unconditionally clear all of the queued messages in a specified message queue.

COMMAND: :Status:MSGCLR <qid|All>

- <**qid**|**All**> [string] mnemonic that references a specific queue. Refer to the mnemonic summary in the introduction of this section. If All is specified then all Message Queues are cleared.

4.11.6 STATUS:STB?

The STatus Byte register performs the same function as the *STB defined in section 3.10 but does not clear the status byte. If it is required to clear the status, then the *STB command should be used.

Query: **:Status:STB?**

→ <Standard Byte Status>END>

☞ **<Standard Byte Status>** NR1 Containing Summary bit status for EQ, MSS, ESB, MAV, RI, SUP, ACP, and SI.

4.11.7 STATUS:TFR[?]

Each Condition Register defined in the controller has a corresponding Transition Filter Register (TFR). Each TFR register has a set of 16 transition filters, one for each corresponding bit of the corresponding CSR. A transition filter select which type of transition of the corresponding condition bit will set the corresponding event bit to TRUE.

There are three possible transition filter behaviors:

Positive (+): A FALSE-to-TRUE transition of the condition bit sets the event bit.

Negative (−): A TRUE-to-FALSE transition of the condition bit sets the event bit.

Either (x): Any transition of the condition bit sets the event bit.

COMMAND: :Status: TFR <register ID>[,<filter16> | <filter1>,<bit>]

- **<register ID>** [String] is a mnemonic that identifies a specific Condition Register.
- **<filter16>** [String] 16 Char({+ | - | x}) is a 16 character string that that defines the behavior of all transition filters associated with the specified Condition Register.
- **<filter1>** [String] {+ | - | x} defines the behavior of one specific transition filter.
- **<bit>** [NRf]/[NR1] {0..15} This optional argument is used to specify the behavior of one transition filter.

EXAMPLE: :Status:TFR Sup1, ++++ - - - - xxxxxxxx

4.12 STATUS – ACUTROL ACT2000 COMMANDS

The Acutrol3000 command language ACL includes commands that are compatible with the status commands of the Acutrol Act2000 IEEE-488 protocol. These commands are summarized in this document for convenience, and are described in their original form in TM-9107F IEEE-488 Programming Manual section 4.4.6 Status Subsystem Command and in Appendix II of the same document.

The tables in the following sections that describe the bit assignments are expanded to identify the bit/function mapping from the corresponding Acutrol3000 status. Not all status has a counter part; consequently, some bits will be identified as not applicable “NA” and set to an inactive default value.

4.12.1 STATUS:ECP

This command is for compatibility with the Act 1000/2000 controllers. ECP stands for Encoder Control Processor and is equivalent in function to the Acutrol3000 Axis Control Processor (ACP).

ACP status will be mapped to the ECP status as shown in the accompanying tables.

COMMAND: **:Status:ECP <axis>,<word>**

→ <ACP status>

- **<axis>** [NR1] specifies which axis which is being configured.
- **<word> = {1 | 2 | 3}(see below)**
  - 1 = ECP Mode
  - 2 = ECP Status
  - 3 = ECP Mode and Status Words

Where the ECP "Mode" is defined as follows:

- 0 = Position Mode, Home Mode
- 1 = Rate Mode, Off Mode
- 2 = ARATe Mode
- 3 = Synthesis Mode
- 4 = Track Mode
- 5 = Abort Mode (ACP internal only)

Figure 9: ECP Status Layout

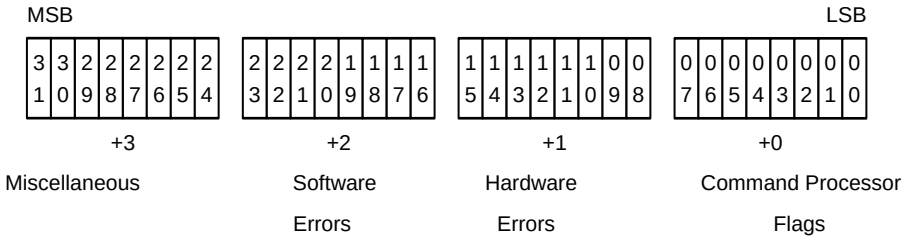


Table 9: ECP Status Bits

Bit	Description	ACP Mapping (TBD)
31	Feedback frozen	FBK Frozen
30	Sector switch 2 active	0
29	Sector switch 1 active	0
28	Power amplifier acknowledge	PA_ACK0
27	Power amplifier fault	PA_FAULT0
26	Power amplifier time-out	0
25	Reserved for future use	0
24	Stow-lock home initialization required	0
23	Track mode data time-out	RT Communication Break
22	Frame time error	Frame Time Error
21	Program error	Error Queue not Empty
20	Miscorrelation error	Miscorrelation Error
19	Non-volatile memory CRC error	AIM Cal Data CRC Error
18	Illegal command from Supervisor received	ECP Command Error
17	CPU fault	?
16	Auto-adjust failure	0
15	Following error trip	Following Error Trip (any axis)
14	Rate trip	Rate Trip (any axis)
13	Reserved for future use	0
12	Self-test failure	Diagnostic Byte Test Error
11	Reserved for future use	0
10	Transducer feedback lost	Loss of FBK (Coarse or Fine)
09	Power supply failure	Diagnostic Byte Test Error
08	Watchdog failure	Watchdog Failure (any)
07	Profile complete	Composite Profile Complete
06	Synthesizer at demanded frequency and amplitude	Synthesizer at frequency and at amplitude
05	Command processor at maximum position	Pos_High_Lim
04	Command processor at minimum position	Pos_Low_Lim
03	Command processor at minimum rate	(-) Rate Lim
02	Command processor at maximum rate	(+) Rate Lim
01	Command processor at minimum acceleration	(-) Accel Lim
00	Command processor at maximum acceleration	(+) Accel Lim

4.12.2 STATUS:MIS

This command is for compatibility with the Act 1000/2000 controllers.

ACP status will be mapped to the Supervisor status as shown in the accompanying tables.

This command now returns the composite status of the ACP (Axis Control Processor) discrete inputs.

Three discrete status <words> are available that indicate the state of discrete inputs, outputs, and the interlock strings. The axis <group> argument selects the range of axes on which to report the discrete signal status.

COMMAND: **:Status:MIS <group>,<word>**

→ <discrete status>

- <group> = **{1 | 2 | …}**
  - 1 = Axes 1 through 3
  - 2 = Axes 4 through 6 etc.
- **<word> = {1 | 2 | 3 | 4} (see below)**
  - 1 = Input Status
  - 2 = Group Status
  - 3 = Output Status
  - 4 = Status Words 1, 2, and 3

Figure 10: MIS Input Status Layout

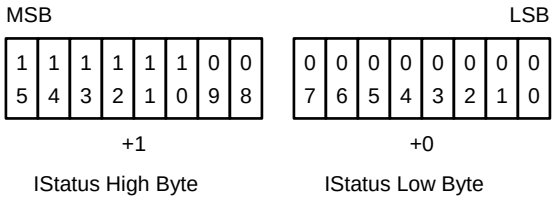


Table 10: MIS Input Status Bits

Bit	Description	ACP Mapping (TBD)
15	Interlock 15, master interlock	Master Interlock(1)
14	Interlock 14, global interlock 3	GLOBAL3
13	Interlock 13, global interlock 2	GLOBAL2
12	Interlock 12, global interlock 1	GLOBAL1
11	Interlock 11, axis 3 switch 4	Axis3, INTLK4
10	Interlock 10, axis 3 switch 3	Axis3, INTLK3
09	Interlock 9, axis 3 switch 2	Axis3, INTLK2
08	Interlock 8, axis 3 switch 1	Axis3, INTLK1
07	Interlock 7, axis 2 switch 4	Axis2, INTLK4
06	Interlock 6, axis 2 switch 3	Axis2, INTLK3
05	Interlock 5, axis 2 switch 2	Axis2, INTLK2
04	Interlock 4, axis 2 switch 1	Axis2, INTLK1
03	Interlock 3, axis 1 switch 4	Axis1, INTLK4
02	Interlock 2, axis 1 switch 3	Axis1, INTLK3
01	Interlock 1, axis 1 switch 2	Axis1, INTLK2
00	Interlock 0, axis 1 switch 1	Axis1, INTLK1

Figure 11: MIS Group Status Layout

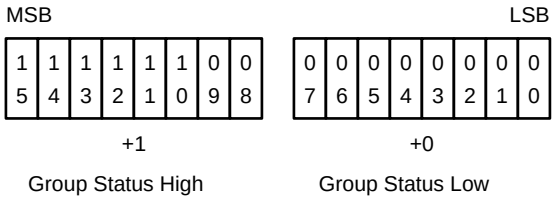


Table 11: MIS Group Status Bits

Bit	Description	ACP Mapping (TBD)
15	Status latched	Interlock Fault
14	Interrupt enabled	AIM Interrupts Enabled
13	Interrupt pending	0
12	Disk drive door unlocked	0
11	MIS driver output enabled	0
10	ECP enable status axis 3	Axis 3 - DE0
09	ECP enable status axis 2	Axis 2 - DE0
08	ECP enable status axis 1	Axis 1 - DE0
07	Front panel disable switch	0
06	Software watchdog	Watchdog Fault (all)
05	ECP go status axis 3	Not Interlock Fault, Axis 3
04	ECP go status axis 2	Not Interlock Fault, Axis 2
03	ECP go status axis 1	Not Interlock Fault, Axis 1
02	Group status axis 3	Not Interlock Fault, Axis 3
01	Group status axis 2	Not Interlock Fault, Axis 2
00	Group status axis 1	Not Interlock Fault, Axis 1

Figure 12: MIS Discrete Output Status Layout

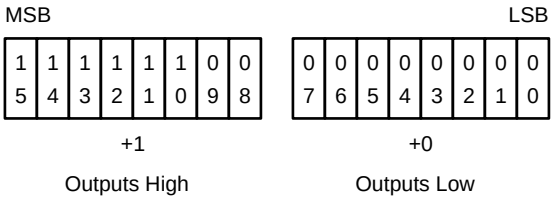


Table 12: MIS Discrete Output Status Bits

Bit	Description	ACP Mapping (TBD)
15	Axis 1 servoed	Axis 1 servoed
14	Axis 2 servoed	Axis 2 servoed
13	Axis 3 servoed	Axis 3 servoed
12	Any axis servoed	Any axis servoed
11	All axes servoed	All axes servoed
10	Addressed closure 1	PROG_OUT1, Axis 1
09	Addressed closure 2	PROG_OUT2, Axis 1
08	Addressed closure 3	PROG_OUT3, Axis 1
07	Addressed closure 4	PROG_OUT4, Axis 1
06	Addressed closure 5	PROG_OUT5, Axis 1
05	Addressed closure 6	PROG_OUT6, Axis 1
04	Remote Mode	Remote Mode
03	Reserved	0
02	Servo fault	Servo Fault (any axis)
01	Power switch override	0
00	Disable light	0

4.12.3 STATUS:SUPERVISOR

This command is for compatibility with the Act 1000/2000 controllers.

ACP status will be mapped to the Supervisor status as shown in the accompanying tables.

The SUPervisor has several 16-bit status <words> that indicate its current state (see below for the bit definitions).

COMMAND: :Status:Supervisor <supervisor status word>

→ <supervisor status word>

☞ <supervisor status word> = **{1 | 2 | 3}** **(see below)**

1 = Word 1

2 = Word 2

3 = Words 1 and 2

EXAMPLE: :STAT:SUP 1

→ #12<2 bytes status word 1>

:S:S 2

→ #12<2 bytes status word 2>

:S:S 3

→ #14<2 bytes status word 1><2 bytes status word 2>

Figure 13: Supervisor Status 1 Layout

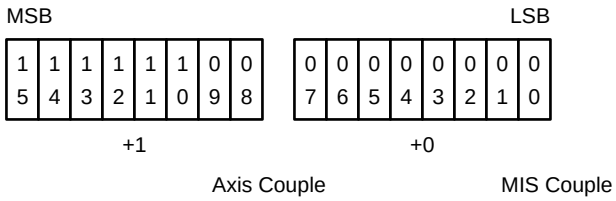


Table 13: Supervisor Status 1 Bits

Bit	Description	ACP Mapping (TBD)
15	An axis time-out error occurred during a mode or servo operation. If enabled, all axes are hard aborted.	??
14	Encoder over or under-run error. If enabled, all axes are hard aborted.	Encoder Error ??
13	An ECP status word caused a soft abort on axis 6	Soft Abort, Axis 6
12	… on axis 5	Soft Abort, Axis 5
11	… on axis 4	Soft Abort, Axis 4
10	… on axis 3	Soft Abort, Axis 3
09	… on axis 2	Soft Abort, Axis 2
08	… on axis 1	Soft Abort, Axis 1
07	Background main loop code execution timed-out. If enabled all axes are hard aborted.	Execution Error ??
06	Bad sync ID received. No axes are aborted.	Bad Sync ID
05	An MIS status word caused a soft abort on axis 6	AIM Hard Abort, Axis 6
04	… on axis 5	AIM Hard Abort, Axis 5
03	… on axis 4	AIM Hard Abort, Axis 4
02	… on axis 3	AIM Hard Abort, Axis 3
01	… on axis 2	AIM Hard Abort, Axis 2
00	… on axis 1	AIM Hard Abort, Axis 1

Figure 14: Supervisor Status 2 Layout

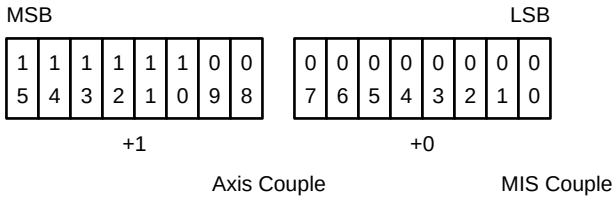


Table 14: Supervisor Status 2 Bits

Bit	Description	ACP Mapping (TBD)
15–02	Internal monitoring or reserved bits	0
01	Prompt/message FIFO full (9 stacked messages).	0
00	Prompt/message FIFO empty (room for 9 new messages).	0

4.13 SYSTEM

The System Subsystem commands provide various system level functions.

4.13.1 SYSTEM:SECURITY[?]

This command :

- BINary. 32-bit signed binary (non-decimal numeric data)
- FLOat. <NR1> format (decimal integer) or <NR2> format (ASCII floating-point with explicit decimal point). This format does not use an exponent.

COMMAND: :Utility:Format {BINary | FLOat}

EXAMPLE: :UTILITY:FORMAT BINARY
:U:F F

The FORMat query returns the currently selected format.

QUERY: :Utility:Format?
→ {BINARY | FLOAT}

EXAMPLE: :U:F?
→ FLOAT

4.14 UTILITY

The UTILity Subsystem commands provide remote control of various system level functions.

4.14.1 UTILITY:FORMAT[?]

This is for compatibility with the ACT2000. All of the demand scaled variables will be configured as either float or binary when the command is received. See :CONFIG:Scaling.

FORMat defines the default formatting for numbers being output from Acutrol. The number format options are:

- BINary. 32-bit signed binary (non-decimal numeric data)
- FLOat. <NR1> format (decimal integer) or <NR2> format (ASCII floating-point with explicit decimal point). This format does not use an exponent.

NOTE: This command can only ne issued by an IEEE-488 interface controller-in-charge.

COMMAND: :Utility:Format {BINary | FLOat}

EXAMPLE: :UTILITY:FORMAT BINARY

:U:F F

The FORMat query returns the currently selected format.

QUERY: :Utility:Format?

→ {BINARY | FLOAT}

EXAMPLE: :U:F?

→ FLOAT

4.14.2 UTILITY:LOCKOUTMODE[?]

This command is used by a remote interface to prevent other interfaced including the Operator interface from issuing commands that would interfere with the motion objectives of the remote computer.

COMMAND: **:Utility:Lockout {ON | OFF}**

EXAMPLE: :U:L ON

The Local Lockout query returns its current state.

QUERY: **:Utility:Lockout?**

→ {ON | OFF}

EXAMPLE: :U:L?

→ OFF

Note: This command is implemented in a rudimentary configuration and will be enhanced in the future to provide more flexible arbitration of which interface is in charge, and to what degree other interfaces are locked out.

4.14.3 UTILITY:MACRO[?]

(Macro sub-command, should it be promoted to a main command?) {Yes, why not}

Let's hold this topic till Phase 2 when we better understand the macros and Python integration.

The MACRo command allows Acutrol macro programs to be loaded, started, and stopped (via the appropriate subcommand). The Command of these macro programs is defined by TM-TBD, Acutrol^{3000} Macro Programming Manual.

To program a macro into Acutrol, the controller must follow this procedure:

1. REMove any existing macro file with the same name.
2. OPEN the macro file.
3. LOAD the text of the macro into the macro file.
4. CLOSe the macro file.
5. The macro file may then be run with the EXECute command.
6. Macro execution may be halted with the STOP command.
7. The contents of a macro file may be read with the UPLoad command.

The MACRo query returns a list of <macro name> strings and <handles> to the variously running macro programs. An empty response string is returned if no macro programs are running.

QUERY: **Utility:Macro?**

→ "<macro name>",<handle>[;"<macro name>",<handle>…]

EXAMPLE: :MACRO?

→ "AUTOSTART",1001;"USER1",1031;"USER2",1321

:M?

→ No macros are running

.

4.14.3.1 UTILITY:MACRO:CLOSE

The MACRo CLOSe command is used to close an OPEN macro file after being LOADed. No error is generated if a CLOSe command is issued without a previous OPEN command.

COMMAND: :UTILity:MACRo:CLOSe

4.14.3.2 UTILITY:MACRO:EXECUTE

The MACRo EXECute command is used to start the execution of a macro file. The <handle> that is returned by this command is formatted as <NR1> data. The value of this <handle> is required if the controller must ever STOP its execution. The <macro name> is case sensitive.

The use of the MACRo query command is not a substitute for saving the value of the <handle>. This is because multiple instances of the same macro may be executed. Each instance must be referred to with a different <handle>.

COMMAND: **:UTILity:MACRo:EXECute "<macro name>"**

→ <handle>

EXAMPLE: :UTIL:MACR:EXEC "TESTMACRO"

→ 31623

:U:M:E "MyMacro"

→ #12<2 bytes of handle data>

4.14.3.3 UTILITY:MACRO:LOAD

The MACRo LOAD command is used to load a macro program into Acutrol. Acutrol will store the program non-volatility. This non-volatile macro program may then be executed.

The controller may then issue multiple LOAD commands to transfer the text of the macro program. Each LOAD command appends the <macro text> to the end of the macro file. Line ends are not automatically added by the LOAD command but must be specified in the <macro text>.

COMMAND: **:UTILity:MACRo:LOAD "<macro text>"**

EXAMPLE:	:U:M:OPEN "TESTMACRO"	OPEN THE MACRO FILE.
	:U:M:LOAD "<MACRO TEXT 1>"	LOAD THE TEXT.
	:U:M:LOAD "<MACRO TEXT 2>"	
	:U:M:LOAD "<MACRO TEXT 3>"	
	:U:M:CLOSE	CLOSE THE FILE.

4.14.3.4 UTILITY:MACRO:OPEN

The MACRo OPEN command is used to open a macro file so that it may be LOADed. The <macro name> is case sensitive.

It is an error to attempt to load a macro program with the same name of an existing file. To allow an existing macro program to be updated, the controller must first REMove the existing macro file then DOWNload the new version.

COMMAND: **:UTILity:MACRo:OPEN "<macro name>"**

EXAMPLE: :UTIL:MACR:OPEN "TESTMACRO"

4.14.3.5 UTILITY:MACRO:REMOVE

The MACRo REMove command is used to delete a macro file from non-volatile storage. The <macro name> is case sensitive.

No error is generated if an attempt to REMove a non-existing file is made.

COMMAND: **:UTILity:MACRo:REMove "<macro name>"**

EXAMPLE: :UTIL:MACR:REMOVE "TESTMACRO"

4.14.3.6 UTILITY:MACRO:STOP

The MACRo STOP command is used to stop the execution of an executing macro. No error is generated if the <handle> does not refer to an executing macro.

COMMAND: **:UTILity:MACRo:STOP <handle>**

EXAMPLE: :UTIL:MACR:STOP 31623

:U:M:S #12<2 bytes of handle data>

4.14.3.7 UTILITY:MACRO:UPLOAD

The MACRo UPLoad command is used to retrieve the text of a macro file. The text is returned one line at a time. The fist UPLoad command must contain the <macro name>. Subsequent UPLoad commands must contain a null <macro name> argument until a null string is returned.

COMMAND: **:UTILity:MACRo:UPLoad "<macro name>"**

→ "<macro text>"

EXAMPLE: :UTIL:MACR:UPL "TestMacro" (Start macro file.)

→ "<macro text line 1>"

:UTIL:MACR:UPL ""

→ "<macro text line 2>"

:UTIL:MACR:UPL ""

→ "<macro text line N>"

:UTIL:MACR:UPL ""

→ "" End of macro file.

4.14.4 UTILITY:REMOTELOCKOUT[?]

COMMAND: **:Utility:RemoteLockout {ON| OFF}**

EXAMPLE: :U:R ON

The Utility Remote Lockout query returns its current state.

QUERY: **:Utility:RemoteLockout?**

→ {ON | OFF}

EXAMPLE: :U:R?

→ OFF

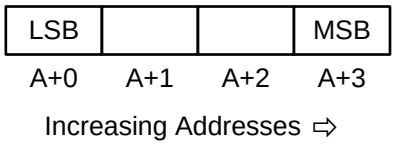
5 APPENDIX A - IEEE-488 CIC SOFTWARE DESIGN CONSIDERATIONS

5.1 DATA FORMATS

5.1.1 SENDING BINARY FORMAT DATA

The Acutrol system is based on Intel microprocessors. The native binary format is "little endian." In "little endian" architectures the data proceeds from low-order byte to high order byte.

A long integer (4 bytes) is represented as:



In order to preserve the performance and architecture of the system the binary data format conforms to this standard.

Binary data may take one of two basic forms:

- Arbitrary length block (#0xxxx). This format is the fastest but only one command may be sent at a time. All data after the "0" until the <END> is considered to be the binary data.
- Definite length arbitrary block #14xxxx. This format indicates the number of length digits to follow. The Acutrol IEEE-488 interface will respond to a maximum of 2 (#2xx) length digits, yielding a maximum of 99 (#299) binary bytes to follow.

5.1.2 BINARY DATA SCALING

Binary position data is 32-bit two's complement number whose most significant bit represents half-scale position. This is usually 180° on a rotational axis.

Binary rate data is a 32-bit two's complement number whose most significant bit represents one thirty-second ($^1/_{32}$) of full-scale position change in one millisecond. This usually translates to a most significant bit weight of 11,250°/s on a rotational axis.

Binary acceleration data is a 32-bit two's complement number whose most significant bit represents one thirty-second ($^1/_{32}$) of full-scale rate change in one millisecond. This usually translates to a most significant bit weight of 703,125°/s^2 on a rotational axis.

Binary jerk data is a 32-bit two's complement number whose most significant bit represents one thirty-second ($^1/_{32}$) of full-scale acceleration change in one millisecond. This usually translates to a most significant bit weight of 43,945,312.5°/s^3 for a rotational axis.

Binary oscillator amplitude data is identical to position data. Negative amplitudes are allowed and will cause the motion to begin in the opposite direction.

Binary oscillator frequency data is a 32-bit two's complement number whose most significant bit represents a frequency of 2000 Hz. Negative frequencies are not allowed.

Binary oscillator phase data is a 32-bit two's complement number whose most significant bit represents 180° of oscillator phase.

Binary time clock data is an unsigned 32-bit number that represents how many milliseconds have passed since the ACP was powered up.

Binary freeze delay data is an unsigned 16-bit number. The value of the number represents how many tenths of microseconds (0.1 µs) the Data Freeze was received after the beginning of the previous frame. This is the age of the feedback in relation to the Data Freeze. Freeze delay data is only valid when the ACP Frozen Status bit is set.

5.1.3 ACP BINARY COMMANDS

The following tables list the binary scaling of the data required for all ACP commands:

- Table 15: ACP Binary Demand Scaling lists the scaling for the DEMand subsystem
- Table 16: ACP Binary Limit Scaling lists the scaling for the LIMit subsystem
- Table 17: ACP Binary Read Scaling lists the scaling for the READ subsystem

Table 15: ACP Binary Demand Scaling

Command	Binary Data Scaling
:DEMand:DELTaposition	Position
:DEMand:POSition	Position
:DEMand:RATE	Rate
:DEMand:ACCeleration	Acceleration
:DEMand:JERK	Jerk
:DEMand:SVECtor	Position followed by Rate
:DEMand:VECTor	Position, followed by Rate, followed by Acceleration
:DEMand:LONGvector	Position, followed by Rate, followed by Acceleration, followed by Jerk
:DEMand:OSCillator	Amplitude, followed by Frequency, followed by Phase
:DEMand:IVARiable	Input variable scaling is the same as its destination

Table 16: ACP Binary Limit Scaling

Command	Binary Data Scaling
:LIMit:LPOSition	Position
:LIMit:HPOSition	Position
:LIMit:RATE	Rate
:LIMit:ACCeleration	Acceleration
:LIMit:JERK	Jerk
:LIMit:VTRip	Rate

Table 17: ACP Binary Read Scaling

Command	Binary Data Scaling
:READ:POSition	Position
:READ:RATE	Rate
:READ:ACCeleration	Acceleration
:READ:JERK	Jerk
:READ:SVECtor	Position followed by Rate
:READ:VECTor	Position followed by Rate followed by acceleration
:READ:OVARiable	Output variable scaling is the same as the variable selected for output
:READ:LOGdata	Logged data scaling is the same as the data that was logged
:READ:TIMEclock	Time clock
:READ:FDELay	Freeze delay

5.1.4 GENERAL GUIDELINES

This section describes some general performance guidelines which apply to both the binary and floating-point formats:

- **Do not re-address the Acutrol to listen on every transfer if sending continuous download demands.**

- **Keep the messages short.**

  Use:

  ```
  :D:P 1,#0xxxx<END>
  ```

  Instead of:

  ```
  :DEMand:POSition 1,#0xxxx<END>
  ```

- **Verbose messages should only be used when speed is not critical.**

- **Monitor data (i.e. READ and STATus subsystems) is collected when Acutrol decodes the incoming message. When the CIC addresses Acutrol to talk that response data is transmitted. If the CIC readdress Acutrol to talk the same collected data is re-transmitted.**

- **Acutrol software is set up to perform all data collection on the same cycle in a single message. (Says Who!)**

- **Use the tree structure for faster data transfers. The Acutrol software has been designed to take advantage of the tree structure documented in this manual.**

  EXAMPLE:

  If it is desired to update the position on two axes, the following sequence is faster:

  ```
  D:P 1,#14xxxx;P 2,#14xxxx<END>
  ```

  Than:

  ```
  :D:P 1,#14xxxx;:D:P 2,#14xxxx<END>
  ```

- **Only one monitor message at a time will be processed for response data.**

  EXAMPLE: `:READ:VECTor 1;VECTor 2;:STATus:MIS 1,3;ECP 1,3;ECP 2,3<END>`

The above message will read the vectors for axis 1 and 2, the complete status of the MIS, and the complete status of both ACPs. This will be transmitted to the CIC when Acutrol is addressed to talk.

If the CIC sends:

```
READ:VECTor 1;V 2;:S:M 1,3;E 1,3;E 2,3<END>
READ:POSition 1<END>
```

Only the position on axis 1 will be transmitted to the CIC when Acutrol is addressed to talk. The second monitor command overwrote the first one.

5.1.5 BINARY FORMAT DATA

To achieve maximum performance over the IEEE-488 bus, binary format should be used to download and monitor. There are several considerations when writing code for the CIC (Controller in Charge):

- **If the software mixes binary and float (ASCII) format, system performance will degrade. This is a consequence of the ASCII conversion and scaling.**

- **If only one download demand is necessary (i.e. position, rate, or acceleration) then send only that demand. <Indefinite Length Arbitrary Block Response> format (#0xxxx) is faster than <Definite Length Arbitrary Response Block Data> format (#14xxxx).**

- **Acutrol software guarantees that all binary download demands begin execution by the ACPs in the same cycle if they are in a single message.**
- **The Acutrol IEEE-488 software is written to transfer download data to the ACPs immediately then have them begin execution on the next frame. It is the CIC's responsibility to time the data transfers. The current encoder frame time is TBD microseconds. If data is sent faster than this, the ACPs will use only the data closest to the frame. The same thing is true of the monitor response data.**

5.1.6 FLOAT FORMAT DATA

Floating-point format should be used when speed is not a consideration. Acutrol will accept floating-point data in the IEEE-488.2 <NRf> format and respond with data in the <NR2> format.

Acutrol will respond slowly (compared to binary data) because of the extensive decoding, scaling, and manipulation necessary to produce this format.

Some design considerations include:

- **The Acutrol software will round to the precision specified in the Setup File. This is true of download and response (monitor) data.**
- **Care must be taken that the units are the same as those in the Setup File. Values (downloaded) that are outside of the full-scale range of a variable will wrap around.**
- **No error checking is performed for multiple decimal points, multiple signs, or multiple exponents.**
- **A single number should be limited to less than 28 ASCII digits (including sign and decimal point). The number will be ignored if greater than this limit.**

5.2 USING A NATIONAL INSTRUMENTS NI-488 DEVICE DRIVER

5.2.1 CONFIGURING THE NI-488 DRIVER

If the CIC (Controller in Charge) is using a National Instruments NI-488 device driver, it should be configured to work with Acutrol. How this is done is dependent upon the particular NI-488 installation. See your NI-488 documentation for assistance.

The following items should to be configured to work optimally with Acutrol. Other NI-488 settings do not seriously affect communication with Acutrol.

Acutrol GPIB device name	set to unique string[2]
Primary GPIB Address	Acutrol ADDRESS parameter
Secondary GPIB Address	NONE
Timeout setting	≥ 1sec
Serial Poll Timeout	≥ 1sec
Terminate Read on EOS	No
Set EOI with EOS on Writes	No
Send EOI at end of Write	Yes
Enable Repeat Addressing	No

[2] Whatever string is entered as the device name will not be available for use as file and directory names.

5.2.2 CHANGING TO REMOTE

Acutrol may be placed into remote using the National Instruments NI-488 driver as follows:

1. Open the Acutrol GPIB device.

   ```
   int hAct = ibfind( "DEV1" );
   ```

   Substitute the appropriate GPIB device name for "DEV1" used to represent Acutrol.
2. Assert REN and address Acutrol to listen by sending a null message.

   ```
   ibwrt( hAct, "", 0 );
   ```

5.2.3 RETURNING TO LOCAL

Acutrol may be returned to local using the National Instruments NI-488 driver as follows:

Send Acutrol to local with the command

```
ibloc( hAct );
```

5.2.4 WRITING TO ACUTROL

Device dependent messages may be sent to Acutrol using the National Instruments NI-488 driver as follows:

```
char szMsg[] = { "device dependent message" };
ibwrt( hAct, szMsg, (sizeof szMsg)–1 );
```

The –1 is required to prevent sending the terminating null character to Acutrol.

5.2.5 READING FROM ACUTROL

Data may be read from Acutrol using the National Instruments NI-488 driver as follows.

Send the device dependent message to Acutrol

```
char szMsg[] = {"device dependent message"};
ibwrt( hAct, szMsg, (sizeof szMsg)–1 );
```

The –1 is required to prevent sending the terminating null character to Acutrol.

Read the data from Acutrol

```
char achBuffer[256];
ibrd( hAct, achBuffer, sizeof achBuffer );
```

The NI-488 variables ibcnt and ibcntl will contain the number of bytes actually input from Acutrol.

5.2.6 SERIAL POLL

A serial poll may be performed using the National Instruments NI-488 driver as follows:

```
char  achPollByte;
ibrsp( hAct, &achPollByte );
```

5.2.7 PARALLEL POLL

A parallel poll may be conducted using the National Instruments NI-488 driver as follows:

1. Configure the Acutrol response to a parallel poll

```
ibppc( hAct, ACUTROL_RESPONSE);
```

2. Conduct a parallel poll

```
char  achPollByte;
ibrpp( hAct, &achPollByte );
```

6 APPENDIX B – BIT STATUS DEFINITIONS

6.1 ERROR QUEUES

TBD

6.2 STATUS MESSAGE QUEUES

TBD

6.3 SYSTEM INTERLOCKS

G1INTLK Global Interlock Status

Bit#	Discrete Status Definition	Bit#	Discrete Status Definition
0	Axis1, INTLK1	8	Axis3, INTLK1
1	Axis1, INTLK2	9	Axis3, INTLK2
2	Axis1, INTLK3	10	Axis3, INTLK3
3	Axis1, INTLK4	11	Axis3, INTLK4
4	Axis2, INTLK1	12	GLOBAL1
5	Axis2, INTLK2	13	GLOBAL2
6	Axis2, INTLK3	14	GLOBAL3
7	Axis2, INTLK4	15	Master Interlock(1)

G2INTLK Global Interlock Status

Bit#	Discrete Status Definition	Bit#	Discrete Status Definition
0	Axis4, INTLK1	8	Axis6, INTLK1
1	Axis4, INTLK2	9	Axis6, INTLK2
2	Axis4, INTLK3	10	Axis6, INTLK3
3	Axis4, INTLK4	11	Axis6, INTLK4
4	Axis5, INTLK1	12	GLOBAL4
5	Axis5, INTLK2	13	GLOBAL5
6	Axis5, INTLK3	14	GLOBAL6
7	Axis5, INTLK4	15	Master Interlock(2)

A(n)INTLK

Bit#	Discrete Status Definition	Bit#	Discrete Status Definition
0	GPD_IN0 Pin	8	INTLK2
1	GPD_IN1 Pin	9	Master Interlock
2	GPD_IN2 Pin	10	AIM Fault – DMA Sync Error
3	GPD_IN3 Pin	11	AIM Fault – Software Watchdog
4	GPD_IN4 Pin	12	AIM Fault – 10K Clock Watchdog
5	GPD_IN5 Pin	13	AIM Fault – Serial Data Link Fault
6	PA_ACK0	14	AIM Fault – Software Interlock Link
7	INTLK1	15	AIM Fault – 10K Lock Fault

A(n)SWFAULT

Bit#	Discrete Status Definition	Bit#	Discrete Status Definition
0	Software Abort Initiated	8	Coarse Transducer FBK Loss
1	Maximum Rate Fault (Rate Trip)	9	Self-test Failure (Byte test summary)
2	Position Tracking Error Fault	10	
3	Frame Time Error	11	
4	Program Error	12	
5	Miscorrelation Error	13	
6	AIM CRC Error (Cal constants)	14	
7	Fine Transducer FBK Loss	15	

6.4 ACP PROCESSES

A(n)STATIC (Bit0)

Bit#	Discrete Status Definition	Bit#	Discrete Status Definition
0	Mode 0 \	8	Status Latched
1	Mode 1 | Binary Coded	9	Data Freeze Active
2	Mode 2 | See Mode[?] section 4.7	10	
3	Mode 3 /	11	
4	Axis Servoed	12	
5	Normal Unservo	13	
6	Software Abort	14	
7	Hardware Abort	15	

A(n)DYN (Bit1)

Bit#	Discrete Status Definition	Bit#	Discrete Status Definition
0	(+)Jerk Limit	8	Position Profile Complete
1	(-)Jerk Limit	9	Rate Profile Complete
2	(+)Acceleration Limit	10	Synthesizer at Amplitude
3	(-)Acceleration Limit	11	Synthesizer at Frequency
4	(+)Rate Limit	12	50% Position Following Error
5	(-)Rate Limit	13	BEMF Limiting
6	Position High Limit	14	Alternate Filters
7	Position Low Limit	15	

A(n)ERR (Bit 2)

Bit#	Discrete Status Definition	Bit#	Discrete Status Definition
0	Floating Point Exception	8	
1	Segmentation Fault	9	
2	Servo Not Enabled	10	
3	DMA Interrupt Not Enabled	11	
4	Divide by Zero	12	
5	Drive Not Enabled	13	

6	NAN Detected	14	
7	General Axis Error	15	

A(n)BITE (Bit 3)

Bit#	Discrete Status Definition	Bit#	Discrete Status Definition
0	BITE Test Bit0	8	BITE Test Bit8
1	BITE Test Bit1	9	BITE Test Bit9
2	BITE Test Bit2	10	BITE Test Bit10
3	BITE Test Bit3	11	BITE Test Bit11
4	BITE Test Bit4	12	BITE Test Bit12
5	BITE Test Bit5	13	BITE Test Bit13
6	BITE Test Bit6	14	BITE Test Bit14
7	BITE Test Bit7	15	BITE Test Bit15

A(n)DIAG (Bit 4)

Bit#	Discrete Status Definition	Bit#	Discrete Status Definition
0	Temperature Fault	8	-12Volt Fault
1	Temperature Warning	9	-12Volt Warning
2	+5Volt Analog Fault	10	+12Volt Fault
3	+5Volt Analog Warning	11	+12Volt Warning
4	Ref2 Fault	12	+5Volt Digital Fault
5	Ref2 Warning	13	+5Volt Digital Warning
6	Ref1 Fault	14	Temperature Fault
7	Ref1 Warning	15	Temperature Warning

A(n)PGOUT (Bit5)

Bit#	Discrete Status Definition	Bit#	Discrete Status Definition
0	Programmable discrete data bit 0	8	Programmable discrete data bit 8
1	…	9	…
2	…	10	…
3	…	11	…
4	…	12	…
5	…	13	…
6	…	14	…
7	Programmable discrete data bit 7	15	Programmable discrete data bit 15

A(n)LOCAL (Bit6): AIM Discrete (Local) Status 4, Direct Register $0E

Bit#	Discrete Status Definition	Bit#	Discrete Status Definition
0	GPD_IN0 Pin	8	DE0
1	GPD_IN1 Pin	9	DE1
2	GPD_IN2 Pin	10	HW_FAULT
3	GPD_IN3 Pin	11	PA_FAULT0
4	GPD_IN4 Pin	12	PA_ACK0

5	GPD_IN5 Pin	13	INTLK1
6	GPD_IN6 Pin	14	INTLK2
7	GPD_IN7 Pin	15	MASTER_INTLK

6.5 AIM REGISTERS

A(n)ID_REG (Bit _): AIM Multiplex Register $07

Release 1 (default) value $4131 “A1”

Bit#	Discrete Status Definition	Bit#	Discrete Status Definition
0	ASCII Code 1.0	8	ASCII Code 2.0
1	ASCII Code 1.1	9	ASCII Code 2.1
2	ASCII Code 1.2	10	ASCII Code 2.2
3	ASCII Code 1.3	11	ASCII Code 2.3
4	ASCII Code 1.4	12	ASCII Code 2.4
5	ASCII Code 1.5	13	ASCII Code 2.5
6	ASCII Code 1.6	14	ASCII Code 2.6
7	ASCII Code 1.7	15	ASCII Code 2.7

A(n)SWITCH (Bit_): AIM Multiplex Register $05

Bit#	Discrete Status Definition	Bit#	Discrete Status Definition
0	AS0	8	DSB
1	AS1	9	DSC
2	AS2	10	DSD
3	AS3	11	DSE
4	SS0	12	DSF
5	SS1	13	DSG
6	SS2	14	FIFO_FULL
7	DSA	15	N.U. => 0

6.6 SUPERVISOR PROCESSES

SUP0 – Supervisor Operational Status

Bit#	Discrete Status Definition	Bit#	Discrete Status Definition
0	Interrupts Enabled (all axes)	8	Data Log in Process
1	Remote Mode	9	Data Log Error
2	Remote Lockout	10	
3	Local Lockout	11	
4	System Initialization Fault	12	Error Queue Overflow
5	Boot Status	13	Error Queue Empty
6	Global Transform Error	14	Message Queue Overflow*
7	Macro Running	15	Message Queue Empty*

* No message queues implemented at this time.

SUP1 – Parser Status

Bit#	Discrete Status Definition	Bit#	Discrete Status Definition
0	Init Error	8	
1		9	
2		10	
3		11	
4		12	
5		13	
6		14	
7		15	

SUP2 – Realtime Thread

Bit#	Discrete Status Definition	Bit#	Discrete Status Definition
0	Init Error	8	Bad Demand Data
1	Shutdown	9	
2	Host Lost	10	
3	Demand Transition Error	11	
4	Monitor Transition Error	12	
5	Time Tracker Error	13	
6	Floating Point Exception	14	Online
7	Segmentation Fault	15	RT Interface Present

SUP15 – Remote Interface Summary

Bit#	Discrete Status Definition	Bit#	Discrete Status Definition
0	ACL Process	8	RFM Process
1	Supervisor Process	9	RT Thread
2	ACP Process	10	
3	TCP Process	11	
4	XML Process	12	

5	TTY Process	13	
6	IEEE-488 Process	14	
7	RS232 Process	15	

6.7 REMOTE INTERFACE SUMMARY STATUS

RI0 – IEEE-488 Interface (IEEE488)

Bit#	Discrete Status Definition	Bit#	Discrete Status Definition
0	Init Error	8	Overflow
1	Shut Down	9	Acknowledge Error
2	Read Error	10	Setup Error
3	Write Error	11	Thread Create Error
4	Queue Send Error	12	
5	Queue Receive Error	13	
6	Queue Sync Error	14	Interface Active
7	Queue Empty	15	Interface Present

RI1 – Parallel Interface (PARALLEL)

Bit#	Discrete Status Definition	Bit#	Discrete Status Definition
0	Init Error	8	Overflow
1		9	
2	Read Error	10	Setup Error
3	Write Error	11	
4		12	
5		13	
6		14	Online
7		15	Interface Present

RI2 – Reflective Memory Interface (RFM)

Bit#	Discrete Status Definition	Bit#	Discrete Status Definition
0	Init Error	8	Overflow
1	Shut Down	9	Acknowledge Error
2	Read Error	10	Setup Error
3	Write Error	11	Thread Create Error
4	Queue Send Error	12	
5	Queue Receive Error	13	
6	Queue Sync Error	14	Interface Active
7	Queue Empty	15	Interface Present

RI3 – TCP/IP Interface (TCPIP)

Bit#	Discrete Status Definition	Bit#	Discrete Status Definition
0	Init Error	8	Socket Fail

1	Shut Down	9	Connect Fail
2	Read Error	10	No TCP Queues
3	Write Error	11	Thread Create Error
4	Queue Send Error	12	
5	Queue Receive Error	13	
6	Queue Sync Error	14	Connected
7	Queue Empty	15	Interface Present