WAVECREST Corporation

Signal Integrity Analyzer 3000

GPIB Programming Guide

200007-00 REV A

WAVECREST Corporation continually engages in research related to product improvement. New material, production methods, and design refinements are introduced into existing products without notice as a routine expression of that philosophy. For this reason, any current *WAVECREST* product may differ in some respect from its published description but will always equal or exceed the original design specifications unless otherwise stated.

This page intentionally left blank.

GPIB INTERFACE GUIDE

This page intentionally left blank.

SECTION 1 - GPIB INTERFACE

1-1 INTRODUCTION TO REMOTE PROGRAMMING OF THE SIA-3000™

You can program the SIA-3000 to:

- Set up the SIA-3000 and start a measurement.
- Return the setup parameters and measurements to the GPIB controller.

Other tasks are accomplished by combining the basic functions.

It is assumed that you are familiar with the usage of the GPIB. If you are not, please consult your GPIB documentation. In particular, you should be familiar with the concepts of selecting an interface, device addressing, interface initialization as well as the command structure and format for programming an instrument over the GPIB.

1-2 SIA-3000 SYNTAX

The mnemonic representing the operation to be performed by the instrument is known as the "command header." There are different types of command headers that are discussed in more detail in the following paragraphs. Commands may be simple or compound. The simple command headers consist of a single mnemonic, while a compound command header contains two or more program mnemonics. The first mnemonic of a compound header selects a subsystem and the last mnemonic selects the desired function within the subsystem. Mnemonics, within a compound message, are separated by colons.

- To execute a simple command, the syntax is:
  <mnemonic><terminator>

  Example: "`:RUN`"

- To execute a simple command with data:
  <mnemonic><separator><data><terminator>

  Example: "`*SAV 1`"

- To execute a single function in a subsystem (a compound command):
  <Subsystem>:<function><separator><data><terminator>

  Example: "`SYSTem:CHANnel 1`"

In addition to the simple and compound command headers, there are also common command headers to control generic functions in the SIA-3000. An example of a common command function is "reset." The syntax for common command headers is:

***<command header><terminator>**

Example: "`*RST`"

Note that no space or other separator is allowed between the asterisk and the command header.

If a command header is immediately followed by a question mark, then the command is a query.

After a query is received, the SIA-3000™ responds by placing a response in the GPIB output queue. The response will stay in the queue until either the controller reads the response or another command is issued by the controller.

The program commands from the controller are case insensitive: either lower or uppercase letters may be used. The SIA-3000 will always respond using upper case. Either the long form (the complete spelling of a command) or the short form (abbreviated spelling) may be used.

The terminator for a message can be a NL (new line, ASCII 10) character, asserting the GPIB EOI (End-Or-Identify) signal or a combination of both. All three ways are equivalent.

It is possible to send multiple commands and queries to different subsystems in the same command by separating each command with a semicolon. Multiple commands may be any combination of compound and simple commands.

1-3 IEEE-488.2 BUS COMMANDS

IEEE-488.2 defines the action of the SIA-3000 for certain bus commands. A device clear (DCL) or selected device clear (SDR) command clears both the input and output buffers. The parser is reset, and any pending commands are cleared.

The group execute trigger (GET) command causes the same action as the RUN/GO command.

The interface clear (IFC) command halts any bus activity. Control is returned to the system controller, and any command in progress is terminated.

1-4 IEEE-488.2 PROTOCOL

The IEEE-488.2 standard defines the overall scheme for communication with the SIA-3000. Please consult the IEEE-488.2 standard for further clarification of the protocol.

The communications subsystem of the SIA-3000 consists of an input buffer and an output buffer.

The input buffer is a memory area where commands and queries from the controller are stored and processed. The input buffer holds 274 characters or bytes of data.

The output buffer is a memory area where data for the controller is stored until read. The output area is large enough to hold 510 characters or bytes of data. Larger blocks of data are handled by breaking the data into a series of blocks smaller than 510 bytes in size.

The SIA-3000's command parser interprets commands from the controller, and determines what action to take in response.

After power up, or after receiving a device clear command, both the input and output buffers are cleared and the parser is reset. The controller and the SIA-3000 communicate by exchanging program and response messages. The controller should always terminate a program message before reading a response from the SIA-3000.

If the controller sends a query message to the SIA-3000, the next message from the controller should be a response message. The controller should read the entire response from the SIA-3000 before sending another query message.

Execution of commands by the SIA-3000 is in the order that the commands are received. This also includes reception of the group execute trigger (GET) bus command. The controller should not send a group execute trigger command in the middle of a program message.

It is possible to send multiple queries in a query message ("compound query") by use of semicolon message separators. The SIA-3000 responses to a multiple query will also be separated by semicolons.

1-4.1 PROTOCOL EXCEPTIONS

If the SIA-3000 is addressed to talk before the controller sent it a query, it will indicate a query error and not transmit any data bytes over the GPIB. If the SIA-3000 has no response because it was unable to execute the query because of an error, the SIA-3000 will not indicate a query error, and waits for the next message from the controller.

If a command error occurs, it is reported to the controller. An example of a command error would be a syntax error or an unrecognized command. A group execute trigger in the middle of a program message is also considered a command error.

If a parameter is out of range, or the current settings of the SIA-3000 will not allow execution of a requested command or query, then an execution error is reported to the controller.

A device-specific error will be reported by the SIA-3000 if it is unable to execute a command for a strictly SIA-3000 dependent reason.

A query error will be reported if the proper protocol for a query is not followed. Query errors include both "unterminated" and "interrupted" conditions.

If the controller attempts to read a response message before the program message has been terminated (an "unterminated" condition), the SIA-3000 reports a query error. The parser is reset, and any response is cleared from the output buffer, without being sent back to the controller.

If the controller fails to read the entire response message, and attempts to send another program message, the SIA-3000 responds with a query error. The unread portion of the response is discarded by the SIA-3000. The program message from the controller is not affected, and will be processed normally by the SIA-3000.

It is possible for the SIA-3000 to become deadlocked in a condition where both the input and output buffers are full. This can occur if the controller sends a very long program message which contains queries that generate a large number of data bytes in response. The SIA-3000 is unable to accept any more program message bytes under this condition, but the controller cannot read any of the response data bytes until the entire program message has been sent to the SIA-3000. If this situation occurs, the SIA-3000 detects the condition, clears the output queue, and discards responses until it reaches the end of the program message. A query error bit is also set under this condition.

This page intentionally left blank.

2-1 SUMMARY OF SIA-3000™ COMMANDS

In addition to the **Common** commands (see section 2.3) defined for all instruments by IEEE-488.2, the instrument subsystem commands used in the SIA-3000 are:

`System` - Controls some basic functions of the SIA-3000.

`Acquire` - Provides access to the parameters for acquiring and storing data.

`Calibrate` - Provides for the selection of different calibrate functions and retrieves data generated by these functions.

`Channel` - Provides access to the parameters associated with the different channels.

`Display` - Provides access to the parameters for controlling how or what information will be displayed.

`Measure` - Selects the measurements to be made.

`Plot` - Provides access to the plot data recorded from a previously called `:ACQ:`<API structure> command.

`Trigger` - Controls the trigger modes and parameters for each trigger mode.

The following legend is used in the instrument subsystem commands:

<n> - Represents any single channel number between 1 and 10 (required)

<a> - Represents any single arming input between ARM1 and ARM10 (required)

(@ <n,m,x,…>|<n:m>) - Represents an optional channel list/range of channels between 1 and 10

- For single channel measurements, valid commands include:

  `:ACQ:ALL PER (@10), :ACQ:ALL PER (@ 1,3,5) and :ACQ:ALL PER (@7:10)`

- For dual channel (parallel) measurements, the ampersand symbol appears between the reference channel and multiple measurement channels. Only one set of parallel measurements can be sent in a single command. For example:

  `:ACQ:ALL TPD++ (@ 1&2,4,5)`

  (TPD ++ measurements on reference channel 1, data channels 2, 4 and 5)

  `:ACQ:ALL TPD++ (@ 2&3:8)`

  (TPD++ measurements on reference channel 2, data channels 3 through 8)

Rules for Using a Channel List or Range:

- If the channel list is absent, the command is executed using the current measurement channel
- The channels must be entered in ascending order
- If the range of channels specified includes an inactive channel, the device will report an error
- If a measurement error occurs on one of the requested channels, values that indicate a bad measurement will be returned/displayed and the device will attempt to measure the remaining channels in the list

Rules for Using the Group (Pseudo-Parallel) Commands:

- To create a group of commands, send the `:SYST:GROUP<n>ON` command, where *n* represents a group between 1 and 20
- Any commands sent after this command will now be queued inside the device as a group until the `:SYST:GROUP<n>OFF` is sent. **Only one group will be queued at a time.**
- When the `:ACQ:GROUP<n>` command is sent, all of the queued commands within that group will be executed in the order they were received. If any of the commands in the group request data to be sent back, the data will be sent back in the order requested
- All of the commands described in this document can be included in a group except for the following:
  - Common and Root commands listed in Sections 2-3 and 2-4
  - `:SYSTem:HEADer`, `:SYSTem:LONGform`, `:SYSTem:COMPatible`, `:SYSTem:ADDRess`, `:SYSTem:ENDian`, `:SYSTem:TEST`, `:SYSTem:GO`, `:SYSTem:NOGO`, `:SYSTem:STROBeCAL` (Section 2-5)
  - `:CALIBRATE` commands (Section 2-7)

2-2 IEEE-488.1 BUS COMMANDS (HARDWARE)

The following commands are IEEE-488.1 bus commands (hardware line ATN true).

Clear Interface (`IFC`) - Halts all bus activity.

Device Clear - The device clear (`DCL`) command causes the device to perform a clear.

Group Execute - Performs the same action as the trigger **`GET`**, **`RUN`** and **`*TRG`** commands. (The device will acquire data.)

2-3 COMMON COMMANDS

The following are common commands defined by IEEE-488.2 and supported by the SIA-3000™.

`*CLS`		Clear Status.
`*ESE`		Event Status Enable.
`*ESE`		Query.
`*ESR`		Event Status Register Query.
`*IDN`		Identification Query.
`*OPC`		Operation Complete.
`*OPC?`		Query.
`*OPT`		Returns the list of instrument options.
`*RCL`	<0-10>	Recall.
`*RST`		Reset. Resets the input and output buffers, resets the parser and clears any pending commands.
`*SAV`	<0-10>	Save.
`*SRE`		Service Request Enable.
`*SRE?`		Query.
`*STB?`		Status Byte Query.
`*TRG`		Causes the SIA-3000 to initiate a measurement.
`*TST?`		Test Instrument Query.

2-4 ROOT COMMANDS

:RUN - Causes the SIA-3000™ to initiate measurement. Does the same function as the *TRG.

:TER? - This query will read the identified TRG Event Register. When the register is read, it is cleared. A one (1) informs the program that the trigger has occurred. Monitor this bit to know when a take sample (burst), pulse find, cable measure or an internal/external calibration is complete.

:LER? - This query will read the Local Event Register. When the query is received and the register is read, it is cleared. A non-zero indicates that a reset is in progress.

:SDS? - This query reads the Special Device Register. When the query is received and the register is read, it is cleared. This register is used to indicate when some commands are complete when they don't set a TRG or MAV bit. Same as bit 3 of a serial poll.

The following is a listing of the commands, organized by the subsystems listed earlier in this document, used to support the SIA-3000™.

2-5 SYSTEM COMMANDS

:SYSTem:ADDRess<0-30>	GPIB address of device
:SYSTem:ARMing/trigger source/trigger sequence/arming channel <a>/arming ref/arming slope/start count/stop count	Arming macro for speed
:SYSTem:ALIAS<ARM1|ARM2|CHAN1|CHAN2><n>	SIA-3000 alias setting for channel or arm
:SYSTem:CHANnel<n>	Set channel
:SYSTem:CHANnel?	Read channel
:SYSTem:COMPatible<ON|OFF>	Enable/disable SIA-3000 Compatibility Mode
:SYSTem:COMPatible?	Read compatibility mode
:SYSTem:DCCHANnel<n>	Select dc measurement channel
:SYSTem:DCCHANnel?	Read dc selected channel
:SYSTem:ELAPsed<OFF|ON>(@ <n,m,x,…>|<n:m>)	Select timed burst mode
:SYSTem:ELAPsed?	Read current timed burst selection
:SYSTem:ENDian<BIG|LITtle>	Select byte swap mode (UNIX)
:SYSTem:ENDian?	Read current byte swap mode
:SYSTem:GATing<ON|OFF>	Turn gating on or off
:SYSTem:GATing?	Read gating selection
:SYSTem:GO	Execute user request
:SYSTem:NOGO	Skip user request
:SYSTem:GROup<1-20><ON|OFF>	Start of stop a command group
:SYSTem:HEADer<OFF|ON>	Select header type
:SYSTem:HEADer?	Read header type selected
:SYSTem:LONGform<OFF|ON>	Select long or short form of headers
:SYSTem:LONGform?	Read long or short-form selected
:SYSTem:MACro/function/trigger source/trigger sequence/percent /start reference voltage/stop reference voltage/start count/stop count	Macro used for speed
:SYSTem:RESET	Perform a system reset (reboots VISI)
:SYSTem:STAT/<ON|OFF>/<AV><JI><MN><MX> (@ <n,m,x,…>|<n:m>)	Selects sets of statistics to be saved
:SYSTem:STAT(@ <n,m,x,…>|<n:m>)?	Reads number of sets acquired
:SYSTem:STRObeARM<n><RISe|FALl>	Select strobe arming input
:SYSTem:STRObeARM<n>?	Read strobe arming input
:SYSTem:STRObeCAL	Initiate a strobe calibration
:SYSTem:STRObeCHANnel<n>	Select strobe input channel
:SYSTem:STRObeCHANnel?	Read strobe channel
:SYSTem:STRObeDELay<value>(@ <n,m,x,…>|<n:m>)	Set strobe delay
:SYSTem:STRObeDELay(@ <n,m,x,…>|<n:m>)?	Read strobe delay
:SYSTem:STRObeINC<value>(@ <n,m,x,…>|<n:m>)	Set step increments|value
:SYSTem:STRObeINC(@ <n,m,x,…>|<n:m>)?	Read step increments|value

:SYSTem:**STRO**be**STAR**t<value>(@ <n,m,x,…>|<n:m>)	Set start delay
:SYSTem:**STRO**be**STAR**t(@ <n,m,x,…>|<n:m>)**?**	Read start delay
:SYSTem:**STRO**be**STEP**s<value>(@ <n,m,x,…>|<n:m>)	Set number of steps
:SYSTem:**STRO**be**STEP**s(@ <n,m,x,…>|<n:m>)**?**	Read number of steps
:SYSTem:**STRO**be**STOP**<value>(@ <n,m,x,…>|<n:m>)	Set stop delay
:SYSTem:**STRO**be**STOP**(@ <n,m,x,…>|<n:m>)**?**	Read stop delay
:SYSTem:**TIME**out<value>	Timeout on pulse measurement. 10 second default (floating-point)
:SYSTem:**TIME**out**?**	Read timeout value
:SYSTem:**WAV**e<PEAK>	Set pulse find to locate peaks
:SYSTem:**WAV**e**?**	Read type of waveform search to be used by pulsefind
:SYSTem:**WIND**ow /start value/stop value/<step increment | **N**umber of steps> (@ <n,m,x,…>|<n:m>)	Set parameters

2-6 ACQUIRE COMMANDS

:ACQuire:**A**djacent**CYC**le (@ <n,m,x,…>|<n:m>)<acyc_header><acyc_structure>	Acquires data in the Wavecrest Api3000's ACYC format
:ACQuire:**ALL**<TT+|TT-|PW+|PW-|PERiod+|PERiod-| TPD+ +| TPD- -| TPD+-|TPD- +|FREQ> (@ <n&m>|<n&m:x>|<n,m,x…)	Select function and return all stats
:ACQuire:**ANAL**ysis**FUNC**tion/Func/LowStartCount/HighStartCount /StopCountDesignator/Increment/DataDes (@ <n&m>|<n&m:x>|<n,m,x…)	Select ftn/chan and return stats
:ACQuire:**ANAL**ysis**JITT**er/Func/LowStartCount/HighStartCount /StopCountDesignator /Increment/DataDes (@ <n&m>|<n&m:x>|<n,m,x…)	Select ftn/chan and return jitter stats
:ACQuire:**ANAL**ysis**RANG**e/Func/LowStartCount/HighStartCount /StopCountDesignator /Increment/DataDes (@ <n&m>|<n&m:x>|<n,m,x…)	Select ftn/chan and return (max-min)/2
:ACQuire:**COMP**lete(@ <n,m,x,…>|<n:m>)**?**	Number of readings taken
:ACQuire:**COUN**t<value>(@ <n,m,x,…>|<n:m>)	Set sample size
:ACQuire:**COUN**t(@ <n,m,x,…>|<n:m>)**?**	Read sample size.
:ACQuire:**DAT**acom(@ <n,m,x,…>|<n:m>)<dcom_header><dcom_structure>	Acquires data in the Wavecrest Api3000's DCOM format
:ACQuire:**DRCG**(@ <n,m,x,…>|<n:m>)<drcg_header><drcg_structure>	Acquires data in the Wavecrest Api3000's DRCG format
:ACQuire:**DUTY**(@ <n,m,x,…>|<n:m>)	Returns duty cycle.
:ACQuire:**EYEH**istogram(@ <n&m>|<n&m:x>)<eyeh_header><eyeh_structure>	Acquires data in the Wavecrest Api3000's EYEH format
:ACQuire:**FUNC**tion<TT+|TT-|PW+|PW-|PERiod+|PERiod- |TPD++|TPD- -|TPD+-|TPD- +|FREQ>	Select function
:ACQuire:**FUNC**tion**?**	Read function selected
:ACQuire:**GROUP**<1-20>	Return all measurements previously requested by the group indicated

:ACQuire:**HIST**ogram(@ <n,m,x,…>|<n:m>)<hist_header><hist_structure>	Acquires data in the Wavecrest Api3000's HIST format
:ACQuire:**JITT**er(@ <n,m,x,…>|<n:m>)<jitt_header><jitt_structure>	Acquires data in the Wavecrest Api3000's JITT format
:ACQuire:**LEV**el(@ <n,m,x,…>|<n:m>)	Pulse find
:ACQuire:**LEV**el(@ <n,m,x,…>|<n:m>)**?**	Perform a Pulse find and read min/max voltage levels
:ACQuire:**LOCK**time(@ <n,m,x,…>|<n:m>)<func_header><func_structure>	Acquires data in the Wavecrest Api3000's FUNC format
:ACQuire:**MEAS**ure(@ <n,m,x,…>|<n:m>)	Takes a measurement and returns average & standard deviation
:ACQuire:**OSC**illoscope(@ <n,m,x,…>|<n:m>)<osci_header><osci_structure>	Acquires data in the Wavecrest Api3000's OSCI format
:ACQuire:**RAND**om**DAT**a(@ <n,m,x,…>|<n:m>)<rand_header><rand_structure>	Acquires data in the Wavecrest Api3000's RAND format
:ACQuire:**RUN**<TT+|TT-|PW+|PW-|PERiod+|PERiod-|TPD++|TPD- -| TPD+-|TPD-+|FREQ> (@ <n&m>|<n&m:x>|<n,m,x…)	Select function, returns average and jitter
:ACQuire:**SET**s**COUN**t<value>(@ <n,m,x,…>|<n:m>)	Set sets size
:ACQuire:**SET**s**COUN**t(@ <n,m,x,…>|<n:m>)**?**	Read sets size
:ACQuire:**STAT**istics(@ <n,m,x,…>|<n:m>)<stat_header><stat_structure>	Acquires data in the Wavecrest Api3000's STAT format
:ACQuire:**TIM**e**DIG**itizer(@ <n,m,x,…>|<n:m>)<tdig_header><tdig_structure>	Acquires data in the Wavecrest Api3000's TDIG format
:ACQuire:**TIM**e**SER**ies(@ <n,m,x,…>|<n:m>)<tser_header><tser_structure>	Acquires data in the Wavecrest Api3000's TSER format
:ACQuire:**WIND**ow/start value/stop value/ <step increment|**N**umber of points> (@ <n,m,x,…>|<n:m>)	Set parameters and return voltage average

2-7 CALIBRATE COMMANDS

:CALibrate:**DATA** <#xy...dddddddddd…>	Set external calibration values
:CALibrate:**DATA?**	Read external calibration values
:CALibrate:**DESKEW**	Initiate channel-channel deskew
:CALibrate:**DESKEWDC**	Initiate channel-channel deskew with DC offset
:CALibrate:**INT**ernal	Initiate internal calibration
:CALibrate:**SIG**nal<"OFF, "10MSQ, "900M OUTP, "900M INP">	Set internal calibration signal
:CALibrate:**SIG**nal**?**	Read cal signal setting
:CALibrate:**STAT**us**?**	Returns current status of cals
:CALibrate:**XINT**ernal<ASCII value>	Initiate extended internal cal
:CALibrate:**XINT**ernal**?**	Read extended cal value

2-8 CHANNEL COMMANDS

:**CHAN**nel<n>:**EXT**ernalarm<a>	Select event external arm
:**CHAN**nel<n>:**EXT**ernalarm**?**	Read selected external arm
:**CHAN**nel<n><**STAR**t|**STOP**>:**LEV**el<value>	Set event trip level
:**CHAN**nel<n><**STAR**t|**STOP**>:**LEV**el**?**	Read event level
:**CHAN**nel<n><**STAR**t|**STOP**>:<MIN|MAX>**?**	Read STARt|STOP min or max level
:**CHAN**nel<n><**STAR**t|**STOP**>:**COUN**t<value>	Set event arm on *n*th count
:**CHAN**nel<n><**STAR**t|**STOP**>:**COUN**t**?**	Read event arm on *n*th count
:**CHAN**nel:**SWIT**ch**IDN?**	Returns Version of DSM-16
:**CHAN**nel:**SWIT**ch<NN>	Select DSM-16 chan. and switch
:**CHAN**nel:**SWIT**ch**?**	Returns selected chan. and switch
:**CHAN**nel:**SWIT**ch<ON|OFF>	Enables/disables DSM-16 front panel switches.

2-9 DISPLAY COMMANDS

:**DISP**lay:**FILT**er<ON|OFF>	Select filtering on or off
:**DISP**lay:**FILT**er**?**	Read selected filtering
:**DISP**lay:**FILT**er<**MIN**imum**|MAX**imum><value>	Set filter value
:**DISP**lay:**FILT**er<**MIN**imum**|MAX**imum>**?**	Read filter value
:**DISP**lay:**LEV**el<value value>	Set percent
:**DISP**lay:**LEV**el**?**	Read pulse percent
:**DISP**lay:**USER**<ON|OFF>	Selects user reference voltages for current function
:**DISP**lay:**USER?**	Reads current user state

2-10 MEASURE COMMANDS

:**MEAS**ure:**AVER**age(@ <n,m,x,…>|<n:m>)**?**	Read average
:**MEAS**ure:**DATA**(@ <n,m,x,…>|<n:m>)**?**	Return measurement data
:**MEAS**ure:**DATA4**(@ <n,m,x,…>|<n:m>)**?**	Return measurement data as floating point
:**MEAS**ure:**DATAT**(@ <n,m,x,…>|<n:m>)**?**	Return timed burst elapsed times
:**MEAS**ure:**DCV**level(@ <n,m,x,…>|<n:m>)**?**	Read dc level
:**MEAS**ure:**MIN**imum(@ <n,m,x,…>|<n:m>)**?**	Read minimum measured value
:**MEAS**ure:**MAX**imum(@ <n,m,x,…>|<n:m>)**?**	Read maximum measured value
:**MEAS**ure:**RANG**e(@ <n,m,x,…>|<n:m>)**?**	Read range of measured values
:**MEAS**ure:**SDEV**iation(@ <n,m,x,…>|<n:m>)**?**	Read standard deviation
:**MEAS**ure:**JITT**er(@ <n,m,x,…>|<n:m>)**?**	Read jitter
:**MEAS**ure:**STAT4**(@ <n,m,x,…>|<n:m>)**?**	Returns statistical data defined by SYST:STAT for number of samples as float
:**MEAS**ure:**STRO**be**VLEV**el(@ <n,m,x,…>|<n:m>)**?**	Read strobed voltage level
:**MEAS**ure:**VMAX**imum(@ <n,m,x,…>|<n:m>)**?**	Read strobed maximum voltage
:**MEAS**ure:**VMIN**imum(@ <n,m,x,…>|<n:m>)**?**	Read strobed minimum voltage

`:MEASure:VSDEViation(@ <n,m,x,…>|<n:m>)?`	Read voltage standard deviation
`:MEASure:VDATA(@ <n,m,x,…>|<n:m>)?`	Return strobed measured points
`:MEASure:VDATA4(@ <n,m,x,…>|<n:m>)?`	Return measurement data as floating point
`:MEASure:WINDow(@ <n,m,x,…>|<n:m>)?`	Return average strobed voltage

2-11 PLOT COMMANDS

`:PLOT:AdjacentCYCle(@n)` <ACYC plot offset value>	Gathers graphical (plot) data acquired from the latest `:ACQuire:ACYC` command for the channel specified
`:PLOT:DATacom(@n)` <DCOM plot offset value>	Gathers graphical (plot) data acquired from the latest `:ACQuire:DATacom` command for the channel specified
`:PLOT:DRCG(@n)` <DRCG plot offset value>	Gathers graphical (plot) data acquired from the latest `:ACQuire:DRCG` command for the channel specified
`:PLOT:EYEHistogram(@n)` <EYEH plot offset value>	Gathers graphical (plot) data acquired from the latest `:ACQuire:EYEHistogram` command for the channel specified
`:PLOT:HISTogram(@n)` <HIST plot offset value>	Gathers graphical (plot) data acquired from the latest `:ACQuire:HISTogram` command for the channel specified
`:PLOT:JITTer(@n)` <JITT plot offset value>	Gathers graphical (plot) data acquired from the latest `:ACQuire:JITTer` command for the channel specified
`:PLOT:LOCKtime(@n)` <FUNC plot offset value>	Gathers graphical (plot) data acquired from the latest `:ACQuire:LOCKtime` command for the channel specified
`:PLOT:OSCilloscope(@n)` <OSCI plot offset value>	Gathers graphical (plot) data acquired from the latest `:ACQuire:OSCilloscope` command for the channel specified
`:PLOT:RANDomDATa(@n)` <RAND plot offset value>	Gathers graphical (plot) data acquired from the latest `:ACQuire:RANDDAT` command for the channel specified
`:PLOT:TIMeDIGitizer(@n)` <TDIG plot offset value>	Gathers graphical (plot) data acquired from the latest `:ACQuire:TIMeDIGitizer` command for the channel specified
`:PLOT:TIMeSERies(@n)` <TSER plot offset value>	Gathers graphical (plot) data acquired from the latest `:ACQuire:TIMeSERies` command for the channel specified

2-12 TRIGGER COMMANDS

`:TRIGger:DELay<step value>`	Set trigger delay in 25ps steps (in terms of steps from –40 to -40, with 0 as the nominal delay value)
`:TRIGger:DELay?`	Read trigger delay setting
`:TRIGger:LEVel<value>`	Set trigger level
`:TRIGger:LEVel<n>?`	Read trigger level setting
`:TRIGger:MAXimum<n>?`	Read maximum arming peak level
`:TRIGger:MINimum<n>?`	Read minimum arming peak level
`:TRIGger:SEQuence<STARt|STOP>`	Selects trigger sequence
`:TRIGger:SEQuence?`	Reads trigger sequence
`:TRIGger:SLOPe<RISe|FALl>`	Set arming direction
`:TRIGger:SLOPe?`	Read arming direction
`:TRIGger:SOURce<EXTernal|AUTomatic>`	Selects trigger source
`:TRIGger:SOURce?`	Read trigger source

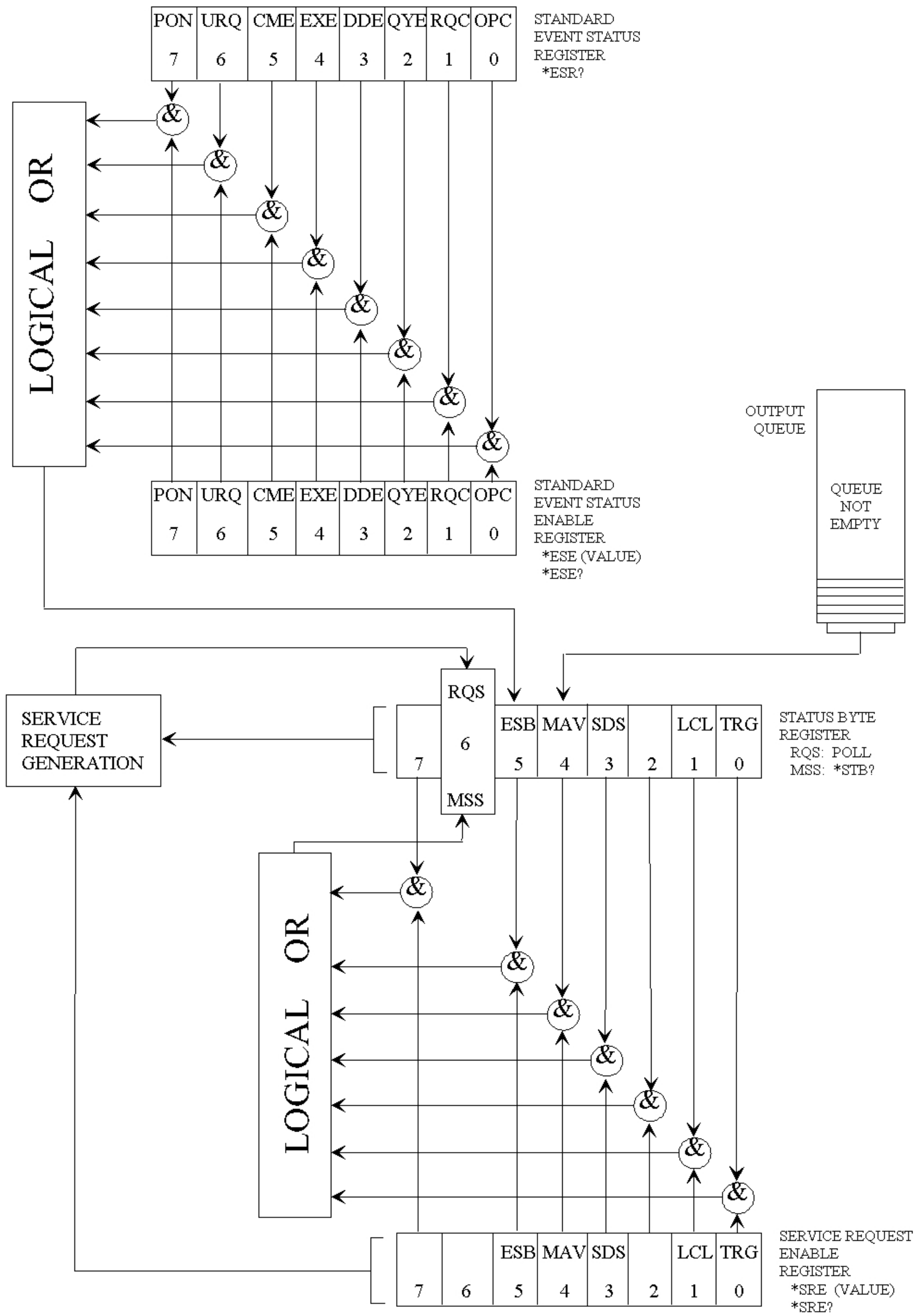
SECTION 3 - COMMON COMMANDS & STATUS

3-1 DESCRIPTION OF THE COMMON COMMANDS & STATUS

IEEE-488.2 defines a set of common commands. These commands perform functions that are common to any type of instrument. They can therefore be implemented in a standard way across a wide variety of instrumentation. All the common commands of IEEE-488.2 begin with an asterisk. There is one key difference between the IEEE-488.2 common commands and the rest of the commands found in this instrument. The IEEE-488.2 common commands do not affect the parser's position within the command tree. Many of these commands are used for status.

Command		Command Name
***CLS**		Clear Status.
***ESE**		Event Status Enable.
***ESE?**		Event Status Enable Query.
***ESR?**		Event Status Register Query.
***IDN?**		Identification Query.
***OPC**		Operation Complete.
***OPC?**		Operation Complete Query.
***OPT**		Returns the list of instrument options.
***RCL**	<0-10>	Recall.
***RST**		Reset. Resets the input and output buffers, resets the parser and clears any pending commands.
***SAV**	<0-10>	Save.
***SRE**		Service Request Enable.
***SRE?**		Service Request Query.
***STB?**		Status Byte Query.
***TRG**		Causes the SIA-3000 to initiate a measurement.
***TST?**		Test Instrument Query.

The bits in the status byte act as summary bits for the data structures residing behind them. In the case of queues, the summary bit is set if the queue is not empty. For registers, the summary bit is set if any enabled bit in the event register is set. The events are enabled via the corresponding event enable register. Events captured by an event register remain set until the register is read or cleared. Registers are read with their associated commands. The "`*CLS`" command clears all event registers and all queues except the output queue. If "`*CLS`" is sent immediately following a <program message terminator>, the output queue will also be cleared.



PON URQ CME EXE DDE QYE RQC OPC
7 6 5 4 3 2 1 0
STANDARD EVENT STATUS REGISTER *ESR?
LOGICAL OR
PON URQ CME EXE DDE QYE RQC OPC
7 6 5 4 3 2 1 0
STANDARD EVENT STATUS ENABLE REGISTER *ESE (VALUE) *ESE?
OUTPUT QUEUE
QUEUE NOT EMPTY
SERVICE REQUEST GENERATION
RQS 6 MSS
7 ESB MAV SDS LCL TRG
5 4 3 2 1 0
STATUS BYTE REGISTER RQS: POLL MSS: *STB?
LOGICAL OR
7 6 ESB MAV SDS LCL TRG
5 4 3 2 1 0
SERVICE REQUEST ENABLE REGISTER *SRE (VALUE) *SRE?

Figure 3-1 STATUS REPORTING

3-1.1 BIT DEFINITIONS

CME - Command error. Indicates whether the parser detected an error.

DDE - Device specific error. Indicates whether the device was unable to complete an operation for device dependent reasons.

ESB - Event status bit. Indicates if any of the conditions in the Standard Event Status Register are set and enabled.

EXE - Execution error. Indicates whether a parameter was out of range, or inconsistent with current settings.

LCL - Indicates whether a remote to local transition has occurred. Indicates when a Device Clear (DCL) is complete.

MAV - Message available. Indicates whether there is a response in the output queue.

MSS - Master summary status. Indicates whether the device has a reason for requesting service. This bit is returned for the `*STB?` query.

OPC - Operation complete. Indicates whether the device has completed all pending operations.

PON - Power on. Always 1.

QYE - Query error. Indicates whether the protocol for queries has been violated.

RQC - Request control. Indicates whether the device is requesting control. Asking for a simulated GO key to be executed.

RQS - Indicates if the device is requesting service. This bit is returned during a serial poll. `RQS` will be set to 0 after being read via a serial poll (MSS is not reset by *STB?).

SDS - Special device status.

TRG - Indicates whether a trigger has been received.

URQ - User request. Indicates whether a front panel key has been pressed.

3-1.2 KEY FEATURES

A few of the most important features of Status Reporting are shown below.

Operation Complete - The IEEE-488.2 structure provides one technique that can be used to find out if any operation is finished. The "OPC" command, when sent to the instrument after the operation of interest, will set the OPC bit in the Standard Event Status Register. If the OPC bit and the RQS bit have been enabled, a service request will be generated.

```
Send(0,5,"*SRE;*ESE1",11,EOI);          !enables an OPC service request.
Send(0,5,"*TRG;*OPC",9,EOI);            !initiates data acquisition.
                                        !will generate a SRQ when the
                                        !acquisition is complete.
```

The Trigger Bit - The TRG bit indicates if the device has received a trigger. The TRG event register will stay set after receiving a trigger until it is cleared by reading it or using the *CLS command. If your application needs to detect multiple triggers, the TRG event register must be cleared after each one.

```
Send(0,5,"*SRE1",6,EOI);          !enables a trigger service request.
                                  !the next trigger will generate an SRQ.
Send(0,5,":TER?",5,EOI);          !queries the TRG event register, thus
Send(0,5,"*TRG",4,EOI);           !clearing it.
                                  !the next trigger can now generate an
Wait SRQ(0,result);               !SRQ.
```

Status Byte - If the device is requesting service (RQS set), and the controller serial polls the device, the RQS bit is cleared. The MSS bit (read with *STB?) will not be cleared by reading it. The status byte is not cleared when read, except for the RQS bit.

Serial Poll - The SIA-3000™ supports the IEEE-488.1 serial poll feature. When a serial poll of the instrument is requested, the RQS bit is returned on bit 6 of the status byte.

Using Serial Poll - This example will show how to use the service request by conducting a serial poll of all instruments on the bus. In this example, assume that there are two instruments on the bus; a DTS at address 5 and a printer at address 1. These address assumptions are made throughout this manual, and it is also assumed that we are operating on GPIB controller board address 0.

The program command for serial poll using IEEE-488.2 in "C" is `ReadStatusByte (0,5,result);`. The address 005 is the address of the SIA-3000 in this example. The command for checking the printer is `ReadStatusByte (0,1,result);` because the address of that instrument is 01 on bus address 0. This command reads the contents of the GPIB Status Register into the variable called result. At that time bit 6 of the variable result can be tested to see if it is set (bit 6=1).

The serial poll operation can be conducted in the following manner.

1. Enable interrupts on the bus. This allows the controller to “see” the SRQ line.
2. If the SRQ line is high (some instrument is requesting service) then check the instrument at address 1 to see if bit 6 of its status register is high.
3. Disable interrupts on the bus.
4. To check whether bit 6 of an instruments status register is high, use the following command line:

```
If (result & 0x40){
  then
  }
```

5. If bit 6 of the instrument at address 1 is not high, then check the instrument at address 5 to see if bit 6 of its status register is high.
6. As soon as the instrument with status bit 6 high is found, check the rest of the status bits to determine what is required.

The `ReadStatusByte (0,5,result);` command causes much more to happen on the bus than simply reading the register. This command clears the bus, automatically addresses the talker and listener, sends `SPE` (serial poll enable) and `SPD` (serial poll disable) bus commands, and reads the data. For more information about serial poll, refer to your controller manual, and programming language reference manuals.

After the serial poll is completed, the RQS bit in the SIA-3000 Status Byte Register will be reset if it was set. Once a bit in the Status Byte Register is set, it will remain set until the status is cleared with a `*CLS` command, or the instrument is reset.

Parallel Poll - The SIA-3000 does not support the parallel poll feature.

3-2 *CLS (Clear Status) command

The `*CLS` (clear status) common command clears the Event Status Register, the Status Byte Register, the trigger bit, the local bit and the error queue.

The Event Status Register is read by the `*ESR?` query. The Status Byte Register is read by the `*STB?` command or a serial poll.

Command syntax- *CLS

Example: `Send(0,5,”*CLS”,4,EOI);`

Query Syntax- None

3-3 *ESE (Event Status Enable) command/query

The *ESE command sets the Standard Event Status Enable Register bits. The Standard Event Status Enable Register contains a mask value for the bits to be enabled in the Standard Event Status Register. A one (1) in the Standard Event Status Enable Register will enable the corresponding bit in the Standard Event Status Register, a zero will disable the bit. Refer to Table 3-1 for information about the Standard Event Status Enable Register bits, bit weights, and what each bit masks.

The *ESE query returns the current contents of the register.

Command Syntax - *ESE <mask>

<mask>::=0 to 255

Example: `Send(0,5,"*ESE 64",7,EOI);`

In this example, the `*ESE 64` command will enable URQ, user request, bit 6 of the Standard Event Status Enable Register. Therefore, when a front-panel key is pressed, the event summary bit (ESB) in the Status Byte Register will also be set.

Event Status Enable Register
(High - Enables the ESR bit)

Bit	Weight	Enables
7	128	PON-Power On
6	64	URQ-User Request
5	32	CME-Command Error
4	16	EXE-Execution Error
3	8	DDE-Device Dependent Error
2	4	QYE-Query Error
1	2	RQC-Request Control
0	1	OPC-Operation Complete

Table 3-1 Standard Event Status Enable Register

Query Syntax - *ESE?

Returned Format: <mask><NL>
<mask>::=0 to 255

Example:
```
Send(0,5,"*ESE?",5,EOI);
Received(0,5,Event,1,EOI);
Printf("%d\n",Event);
```

3-4 *ESR? (Event Status Register) query

This ***ESR** query returns the contents of the Standard Event Status Register.

NOTE: Reading the register clears the Standard Event Status Register and the ESB bit in the STB register.

Query Syntax: ***ESR?**

Returned Format: <status><NL>
<status>::=0 to 255

Example:
```
Send(0,5,"*ESR?",5,EOI);
Receive(0,5,Event,1,EOI);
Printf("%d\n",Event);
```

With the example (`*ESE=64`), if a front-panel key has been pressed, the variable "event" will contain 64, the URQ (User Request bit).

Table 3-2 shows the Standard Event Status Register. The table shows each bit in the Standard Event Status Register as well as the bit weight. When you read Standard Event Status Register, the value returned is the total bit weights of all bits that are high at the time you read the byte.

Event Status Register

Bit	Bit Weight	Bit Name	Condition
7	128	PON	0=not used-always zero
6	64	URQ	0=no front panel key has been pressed 1=front panel key has been pressed
5	32	CME	0=no command errors 1=a command error has been detected
4	16	EXE	0=no execution error 1=an execution error has been detected
3	8	DDE	0=no device dependent errors 1=a device dependent error has been detected
2	4	QYE	0-no query errors 1=a query error has been detected
1	2	RQC	0=request control
0	1	OPC	0=operation is not complete 1=operation is complete

0 = False = Low
1 = True = High

Table 3-2 Standard Event Status Register

3-5 ***IDN?** (Identification Number) query

The ***IDN?** query allows the instrument to identify itself. It returns the string:

> "WAVECREST, SIA-3000, VERSION MAJOR, VERSION MINOR, REVISION LEVEL."
> VERSION MAJOR = Major version of software release.
> VERSION MINOR = Minor version of software release.
> REVISION LEVEL = Updates to current software release.

An ***IDN?** query must be the last query in a message. Any queries after the ***IDN?** in this program message will be ignored.

Query Syntax- *IDN?

Returned Format: `WAVECREST, SIA-3000, v NN.NN.NN`

Example:
```
CHAR MESSAGE[50];
Send(0,5,"*IDN?",5,EOI);
Receive(0,5,MESSAGE,50,EOI);
Printf("%s\n",MESSAGE);
```

3-6 *OPC (Operation Complete) command/query

The ***OPC** (operation complete) command will cause the instrument to set the operation complete bit in the Standard Event Status Register when all pending device operations have finished. The `*OPC?` query places an ASCII "1" in the output queue when all pending device operations have finished.

Command Syntax- *OPC

Example: `Send(0,5,"*OPC",4,EOI);`

Query Syntax- *OPC?

Example:
```
Send(0,5,"*OPC?",5,EOI);
Receive(0,5,data,1,EOI);
Returned format: "1"
```

3-7 *RCL (Recall) command

The ***RCL** command restores the state of the SIA-3000 from a specified set of saved setups. There can be ten (10) different setups (1 through 10).

Command Syntax- *RCL<specific setup>#

Example: `Send(0,5,"*RCL1",6.EOI);`

Query Syntax- None

NOTE: See common command ***SAV** for specific information recalled/saved.

3-8 *RST (Reset) command

The **`*RST`** command place the instrument in a known state. The output buffer is cleared as well as the ESR and serial poll status registers. Use the interface clear (`IFC`) bus command to perform a hardware reset.

Command Syntax- `*RST`

Example:
```
int result;
Send(0,5,”*CLS”,4,EOI);
Send(0,5,”*RST;*OPC”,9,EOI);
result=0
while ((result&0X20 !=0){ /*wait for reset to finish*/
   ReadStatusByte(0,5,&result);
   }
/*reset complete*/
```

Query Syntax- None

3-9 *SAV (Save) command

The **`*SAV`** command stores the current settings of the SIA-3000 in non-volatile memory. This setup is saved and recalled by specifying a specific setup from 1 to 10. See the list below for the parameters saved. Notice that for each setting (1-10), each of the ten (10) functions has a number of settings saved.

Command Syntax- `*SAV`<specific setup>#

Example: `Send(0,5,”*SAV6”,5,EOI);`

Query Syntax- None

During a SAVE or RECALL, the following parameters are saved for later recall or recalled and used as SIA-3000 parameters:

Arming Source
Filter maximum DC Channel
Filter minimum Strobe delay
Filter On/Off Strobe input channel
Function Selection (defines edge direction)
- Channel selection (Ch1/Ch2/…/Ch*n*)
- Arming event arming sequence
- Start reference voltage
- Stop reference voltage
- External Arm reference voltage
- External Arm edge direction
- Pulse find levels (percentages)
- Start/Stop edge (rising or falling)
- Start/Stop arm on *n*th count

Gating on/off
Sample size
Sets size
Start/Stop external arming inputs
Start/Stop VOH (max peak) voltage
Start/Stop VOL (min peak) voltage
Strobe arming channel
Strobe increment value
Strobe number of points
Strobe start point
Strobe stop point

Notes: The external calibration values are not saved on a SAVE.

3-10 *SRE (Service Request Enable) command/query

The ***SRE** command sets the Service Request Enable Register bits. The Service Request Enable Register contains a mask value for the bits to be enabled in the Status Byte Register. A one in the Service Request Enable Register will enable the corresponding bit in the Status Byte Register, a zero will disable the bit. Refer to table 3-3 for the bits in the Service Requst Enable Register and what they mask.

The ***SRE** query returns the current value.

Command Syntax- *SRE <mask>

<mask>::=0 to 255

Example: `Send(0,5,”*SRE16",7,EOI);`

NOTE: This example enables a service request to be generated when a message is available in the output queue. When a message is available, the MAV bit will be high.

Query Syntax- *SRE?

Returned Format: <mask><NL>

<mask>::=sum of all bits that are set - 0 through 255

Example:

```
Send(0,5,”*SRE?”,5,EOI);
Receive(0,5,ENABLE,1,EOI);
Printf(“%d\n”,ENABLE);
```

Event Status Enable Register
(High - Enables the ESR bit)

Bit	Weight	Enables
7	128	not used
6	64	RQS-Request Service
5	32	ESR-Event Status Register
4	16	MAV-Message Available
3	8	SDS-Sub-Device Status
2	4	MSG-Message - Not Used
1	2	LCL-Local
0	1	TRG-Trigger

Table 3-3 Standard Event Status Enable Register

3-11 **`*STB?`** (Status Byte) query

The **`*STB`** query returns the current value of the instrument's status byte. The **MSS** (Master Summary Status) bit and not **RQS** is reported on bit 6. The **MSS** indicates whether or not the device has at least one reason for requesting service. Refer to table 3-4 for the meaning of the bits in the status byte.

Note: To read the instrument's status byte with RQS reported on bit 6, use the GPIB Serial Poll.

Command Syntax- None

Query Syntax- `*STB?`

Returned Format: <value><NL>
<value>::= 0 through 255

Example:
```
Send(0,5,”*STB?”,5,EOI);
Receive(0,5,STATUS,1,EOI);
Printf(“%d\n”,STATUS);
```

Bit	Bit Weight	Bit Name	Condition
7	128	-------	0=not used
6	64	RQS/MSS	0=instrument has no reason for service 1=instrument is requesting service
5	32	ESR	0=no event status conditions have occurred 1=an enabled event status condition has occurred
4	16	MAV	0=no output messages are ready 1=an output message is ready
3	8	SDS	0=special device status
2	4	MSG Not Used	0=no message has been displayed 1=message has been displayed
1	2	LCL	0=a remote to local transition has not occurred 1=a remote to local transition has occurred
0	1	TRG	0=no trigger has occurred 1=a trigger has occurred

Table 3-4 Status Byte Register

3-12 **`*TRG`** (Trigger Event Register) command

The **`*TRG`** command initiates the DTS to take a measurement. This is the same effect as a Group Execute Trigger (GET) or sending the root command **`RUN`**. Use the root query, **`:TER?`**, to indicate when a measurement is complete.

Command Syntax- `*TRG`

Example:
```
int result, event_status;
Send (0,5,”:TER?”,5,EOI); /*clears the TRG Event Register*/
result = 0
while((result & 0x01) !=0){
   ReadStatusByte(0,5,& result);
   }
Send(0,5,”*CLS”,4,EOI);
Send(0,5,”*TRG”,4,EOI);
while((result & 0x01) !=1){ /*wait for TRG bit of serial poll*/
   ReadStatusByte(0,5,& result);
   }
event_status = 0;
if((result & ESB) = = 1) /*if ESB set*/
   {
   Send(0,5,”*ESR?”,5,EOI);
   Receive(0,5,event_status,1,EOI);
   if((event_status & DDE) !=0) /*if measurement bad*/
   Printf(“failed measurement”);
   }
```

Query Syntax- None

3-13 **`*TST?`** (Test Instrument) query

The **`*TST?`** query initiates a series of tests to be executed.

Command Syntax- None

Returned value:	0 = passed
	Non-zero = failed

Query syntax- `*TST?`

Example:
```
Send(0,5,”*TST?”,5,EOI);
Receive(0,5,status,1,EOI);
```

4-1 DESCRIPTION OF THE ROOT COMMANDS

The **ROOT** commands are used to do a few basic instrument functions or read status.

Root commands:	**:LER?**	**:OPEE?**	**:OPEE**
	:SDS?	**:RUN**	**:TER?**

4-2 LER?

The **LER?** query reads the Local Event Register. When the query is received and the register is read, it is also cleared. The status of the Local Event Register (0 or 1) is indicated by a serial poll status bit 1. When the LCL bit of a serial poll is a 1, the Device Clear (DCL) is complete. See the common command ***RST** for use with the **LER?** query.

Command syntax - None

Query syntax - :LER?

Example:
```
int result;
Send(0,5,":LER?",5,EOI);
ReadStatusByte(0,5,& result);
Printf("%d\n",result);
```

4-3 RUN

The **RUN** command initiates a measurement to be started in the SIA-3000. Performs the same function as common command ***TRG**.

Command syntax - :RUN

Example:
```
Send(0,5,":RUN",4,EOI);
```

Query syntax- None

4-4 SDS?

The **SDS?** query reads the Special Device Status register. When the query is received the register value is returned and the register is cleared. The status of the Special Device Status register (0 or 1) is indicated by a serial poll or STB command on bit 3. This bit is used differently by specific instrument commands.

Recall storage	1 = command complete
Display panel ON	1 = command complete

Command syntax- None

Query syntax- `:SDS?`

Example:
```
int result
Send(0,5,”:SDS?”,5,EOI);
result = 1;
  while((result&0x08) !=0) {
  ReadStatusByte(0,5,& result) ;
  }
Send(0,5,”*RCL5”,5,EOI);
result = 0,
  while((result&0x08 = =0) {
  ReadStatusByte(0,5,& result);
  }
/*command complete*/
```

4-5 TER?

The **TER?** query enables the TRG Event Register to be read. Once the TRG Event Register is read, it is cleared. A one (1) indicates a trigger has occurred. A zero (0) indicates a trigger has not occurred.

Command syntax- None

Query syntax- `:TER?`

Returned Format: Bit 1 of a serial poll will indicate the value of the TRG Event Register.

Example:
```
int result;
Send(0,5,”:TER?”,5,EOI); /*clear TRG bit*/
while((result & 0x01) !=0){
  ReadStatusByte(0,5, & result);
  }
Send(0,5,”*TRG”,4,EOI);
while((result & 0x01) !=1){
  ReadStatusByte(0,5, & result);
  }
/*command complete*/
```

Use the TER query to indicate when the following commands are complete:

Burst (*TRG)
Pulse Finder (:ACQ:LEV)
Internal Calibration
External Calibration
Strobe Calibration
Cable Measure

SECTION 5 - SYSTEM COMMANDS

5-1 DESCRIPTION OF SYSTEM COMMANDS

The **SYSTEM** commands control the way channels are selected, messages are formatted, front panel keys are simulated and how voltage measurement will be taken.

:SYSTem:<command syntax>

ADDRess	**GAT**ing	**STAT**istics	**STRO**be**STEP**s
ALIAS	**GO**	**STRO**be**ARM**	**STRO**be**STOP**
ARMing	**GRO**up	**STRO**be**CAL**	**TIM**eout
CHANnel	**HEAD**er	**STROB**e**CHAN**nel	**WAV**e
COMPatible	**LONG**form	**STRO**be**DEL**ay	**WIND**ow
DCCHANnel	**MAC**ro	**STRO**be**INCR**ement	
ELAPsed	**NOGO**	**STRO**be**INCR**ement	
ENDian	**RESET**	**STRO**be**STAR**t	

5-2 ADDRESS

The **ADDRESS** command permitss the user to change the address assigned to the SIA-3000 when it is communicating over GPIB. For example, the default address for the SIA-3000 is 5, but a user may have already connected an oscilloscope to his/her test system that has the same address. The user could then change the address of the SIA-3000 to any number between 0 and 30 (except 5!) so that a host computer could communicate to both the 3000 and the oscilloscope at the same time.

NOTE: Once the user changes the GPIB address of the SIA-3000 using `:SYST:ADDRess`, they need to follow the call with a system reset (`:SYSTem:RESET`) command in order for the change to take effect. (See `:SYSTem:RESET` command.)

Command syntax - **:SYSTem:ADDR**ess<0-30>

Example: `Send(0,5,":SYSTem:ADDRess5",16,EOI);`

Query syntax- **:SYSTem:ADDR**ess**?**

5-3 ALIAS

The **ALIAS** command is used in DTS Compatibility mode. The **ALIAS** command permits the user to use the legacy DTS set of GPIB commands that reference CHAN1, CHAN2, ARM1 and ARM2. For example, an operator connects signals to channels 3 and 4, and external arming inputs to channels 5 and 6. To use the DTS Compatible set of commands, the user would send `:SYSTem:ALIASCHAN1 3` and `:SYSTem:ALIASCHAN2 4` (Channel 3 is "acting" as DTS Channel 1, Channel 4 is "acting" as DTS Channel 2), then `:SYSTem:ALIASARM1 5` and `:SYSTem:ALIASARM2 6` (Channel 5 is "acting" as DTS ARM1, Channel 6 is "acting" as DTS ARM2)

Command syntax - **:SYST**em**:ALIAS**<ARM1|ARM2|CHAN1|CHAN2><n>

Example: `Send(0,5,":SYSTem:ALIASCHAN1 3",16,EOI);`

Query syntax- None

5-4 ARMING

The **`ARMING`** command is a macro command to allow the sending of all commands related to arming the instrument in one command.

The parameters that can be sent are:

Trigger source	`EXTernal/AUTomatic`
Trigger sequence	`STARt/STOP`
Arming channel input	`<a>`
Arming reference	`±1.1`
Arming slope (edge)	`RISe/FALl`
Start arm on count	`1 to 131072`
Stop arm on count	`1 to 131072`

The parameter's position is defined by a forward slash (/). If a parameter is not being set the forward slash must be used.

Command syntax - `:SYSTem:ARMing/trigger source/trigger sequence /arming channel<a>/arming ref/arming slope/start count/stop count`

Example 1: `Send(0,5,":SYSTem:ARMing/EXT/STAR/2/+0.0001 /RIS/2/256",43,EOI);`

The following example only sets the arming reference voltage and slope.

Example 2: `Send(0,5,":SYSTem:ARMing/ / / /+0/FAL/ /",26,EOI);`

Query Syntax- None

5-5 COMPATIBLE

The **`COMPATIBLE`** command permits the operator to use the DTS Compatible set of GPIB commands when the `:SYSTem:COMPatible ON` command is sent. If the operator wants to switch and use the new SIA-3000 GPIB command set, the operator would send the `:SYSTem:COMPatible OFF` command.

Command syntax - `:SYSTem:COMPatible<ON|OFF>`

Example: `Send(0,5,":SYSTem:COMPatible ON",21,EOI);`

Query syntax - `:SYSTem:COMPatible?`

5-6 CHANNEL

The **`CHANNEL`** command selects the input channel that will be measured.

The **`CHANNEL`** query returns the presently selected channel.

Command syntax - `:SYSTem:CHANnel<n>`

Example: `Send(0,5,":SYSTem:CHANnel1",16,EOI);`

Query syntax - `:SYSTem:CHANnel?`

Example: `Send(0,5,":SYSTem:CHANnel?",16,EOI);`
Response: `<"n">`

5-7 DCCHANNEL

The **`DCCHANNEL`** command selects a DC measurement and the input channel that will be measured.

The **`DCCHANNEL`** query returns the channel presently selected.

Command syntax - `:SYSTem:DCCHANnel<n>`

Example: `Send(0,5,”:SYSTem:DCCHANnel1",18,EOI);`

Query syntax - `:SYSTem:DCCHANnel?`

Example: `Send(0,5,”:SYSTem:DCCHANnel?”,18,EOI);`
Response: `<”n”>`

5-8 ELAPSED

The **`ELAPSED`** command enables the elapsed time counter to be initialized and it will be started when the proper edge gate is received on the designated ARM channel input(s).

Command syntax- `:SYSTem:ELAPsed<OFF|ON>(@<n,m,x,…>|<n:m>)`

Example: `Send(0,5”:SYSTem:ELAPsedON”,16,EOI);`

Query syntax- `:SYSTem:ELAPsed?`

Example: `Send(0,5,”:SYSTem:ELAPsed?”,15,EOI);`
Response: `<”ON”|”OFF”>`

5-9 ENDIAN

The **`ENDIAN`** command is only applicable for operators who use UNIX to communicate with the `SIA-3000` over GPIB. In UNIX systems, numerical data is packaged the opposite of Windows 98 (which the `SIA-3000` uses). In order for UNIX users to receive numerical data in a format they can understand, “byte-swapping” of the data must be performed.

If the user sends the `:SYSTem:ENDian BIG` command, the `SIA-3000` will perform “byte-swapping” on all numerical data before sending it back to the user. To return to regular data packaging, the user would send the `:SYSTem:ENDian LITtle` command.

Command syntax- `:SYSTem:ENDian<BIG|LITtle>`

Example: `Send(0,5”:SYSTem:ENDian BIG”,17,EOI);`

Query syntax- `:SYSTem:ENDian?`

Response: `<”BIG”>`

5-10 GATING

The **`GATING`** command turns gating mode on or off. The selection of gating excludes the use of the current ARM input. When gating is selected, the current ARM edge and reference voltage is associated with gating.

The **`GATING`** query returns the present setting of gating.

Command syntax- `:`**`SYST`**`em:`**`GAT`**`ing<ON|OFF>`

Example: `Send(0,5,”:SYSTem:GATingON”,16,EOI);`

Query syntax- `:`**`SYST`**`em:`**`GAT`**`ing`**`?`**

Example: `Send(0,5,”:SYSTem:GATing?”,15,EOI);`
Response: `<ON|OFF>`

5-11 GO

The **GO** command simulates the user responding to a request for input from the `SIA-3000` front panel. This command would be used in conjunction with two (2) status bits of the Event Status Register (`*ESR?`). The host would look for the event status register bit 1, Request Control (asking for the GO key to be pressed). The host would then send the system go command and wait for the event status register bit 6, User Request, to be set to a one (1) indicating the simulated response from the user was completed.

Command syntax- `:`**`SYST`**`em:`**`GO`**

Example: `Send(0,5,”:SYSTem:GO”,10,EOI);`

Query syntax- None

5-12 GROUP

The **`GROUP`** command permits the user to place the `SIA-3000` in GROUP mode (`:SYSTem:GROUP ON`). When the `SIA-3000` is in GROUP mode, any GPIB commands it receives are recorded but not executed (no measurements are made). When the user turns GROUP mode OFF and sends the `:ACQuire:GROUP<n>` command, all the commands that where recorded earlier are executed automatically without any additional input required from the user. The advantage to this method is that the user can instruct the instrument to perform a series of lengthy and complicated measurements before the measurements are actually made, then simply wait for the data at the end. This results in a shorter execution time then if the user asked for the first measurement, waited for the measurement to finish, retrieve the data, ask for the second measurement, wait, etc., etc.

Command syntax - `:`**`SYST`**`em:`**`GRO`**`up<1-20><ON|OFF>`

Example: `Send(0,5,”:SYSTem:GROUP ON”,15,EOI);`

Query syntax- None

5-13 HEADER

The **HEADER** command allow the option of not having the header returned on a response from the instrument.

The **HEADER** query returns the type of header presently selected.

Command syntax- `:SYSTem:HEADer<OFF|ON>`

Example: `Send(0,5,”:SYSTem:HEADerOFF”,17,EOI);`

Query syntax- `:SYSTem:HEADer?`

Example: `Send(0,5,”:SYSTem:HEADer?”,15,EOI);`
Response: `<”0”|”1”> (OFF or ON)`

5-14 LONGFORM

The **LONGFORM** command selects whether a header is returned from the instrument is of a long form or short form. This command works with the **HEADER** command.

The **LONGFORM** query returns the presently selected long or short form.

Command syntax- `:SYSTem:LONGform<OFF|ON>`

Example: `Send(0,5,”:SYSTem:LONGformOFF”,19,EOI);`

Query syntax- `:SYSTem:LONGform?`

Example: `Send(0,5,”:SYSTem:LONGform?”,17,EOI);`

5-15 MACRO

The **MACRO** command can be used to send multiple commands for a few settings that usually change frequently.

The parameters that can be sent are:

Function	TPD++/TPD—/TPD+-/TPD-+/TT+/TT-/PW+/PW-/PER/FREQ
Trigger Source	EXT/AUT
Arming Enable Sequence	STAR/STOP
Peak Percentage	50 50/80 20/20 80/90 10/10 90

Note: Any combination greater than zero (0) and less than 100 is valid over the GPIB interface

Start Input Voltage Reference	±1.1
Stop Input Voltage Reference	±1.1
Start Count	1 to 131072
Stop Count	1 to 131072

Command syntax- `:SYSTem:MACro/Function/Trigger source/Trigger sequence /percent/start reference voltage/stop reference voltage/start count /stop count`

Example:
```
Send(0,5,”:SYSTem:MACro/TT+/AUT/STOP/80 20/+0.003
          /-0.001/2/256",51,EOI);
```

If a parameter is not used, that location can be left blank.

Example: `Send(0,5,”:SYST:MAC/TPD++/ / / / / / /”,25,EOI);`

Query syntax- None

5-16 NOGO

The **NOGO** command simulates a user response to skip an operation after a request for input from the front panel. This command would be used in conjunction of two (2) status bits of the Event Status Register (`*ESR?`).

The host would look for the event status register bit 1, Request Control (asking for the GO key to be pressed). The host would then send the system nogo command and wait for the event status register bit 6, User Request to be set to a one (1) indicating the simulated pressing of the go key was completed.

Command syntax- **`:SYST`**`em`**`:NOGO`**

Example: `Send(0,5,":SYSTem:NOGO",12,EOI);`

Query syntax- None

5-17 RESET

The **RESET** command reboots *VISI; VISI* closes and restarts again automatically. This also happens when the user presses the HW Reset button in the *VISI* Configuration screen.

Command syntax- **`:SYST`**`em`**`:RESET`**

Example: `Send(0,5,":SYST:RESET",11,EOI);`

Query syntax- None

5-18 STAT (Statistics)

The **STAT** command saves a selected group of statistics for each measurement: Average, Jitter, Minimum and Maximum for the desired number of channels.

The **STAT** query returns the selected group of statistics in ASCII form in the same order every time regardless of what order they were selected. The order is AV, JI, MN, MX.

Command syntax- `:SYSTem:STAT/<ON|OFF>/<AV><JI><MN><MX>(@<n,m,x,…>|<n:m>)`

Example: `Send(0,5,":SYST:STAT/ON/JIAVMXMN",22,EOI);`

Query syntax- `:SYSTem:STAT(@ <n,m,x,…>|<n:m>)?`

Example: `Send(0,5,":SYST:STAT?",11,EOI);`
Response: `<ON|OFF><AV><JI><MN><MX>`

5-19 STROBEARM

The **`STROBEARM`** command selects how a voltage measurement is taken. and selects the signal and edge controlling a strobed voltage measurement. The strobed point on a waveform can be controlled by moving the strobe signal, when not using the signal being strobed, or use the **`STROBEDELAY`** command.

If the strobe arm is not selected, the default “DC measurement without strobing” is used.

The **`STROBEARM`** query returns the strobe arm selected or DC if strobing is not selected.

Command syntax- **`:SYSTem:STRObeARM<n><RISe|FALl>`**

Example: `Send(0,5,”:SYSTem:STRObe3RISe",19,EOI);`

Query syntax- **`:SYST`**`em`**`:STRO`**`be`**`ARM<n>?`**

Example: `Send(0,5,”:SYSTem:STRObe3?”,16,EOI);`

Response: <n>

5-20 STROBECAL

The **`STRObeCAL`** command initiates an Oscilloscope Strobe calibration.

Command syntax- **`:SYSTem:STRObeCAL`**

Example: `Send(0,5,”:SYSTem:STRObeCAL”,17,EOI);`

Query syntax- None

5-21 STROBECHANNEL

The **`STROBECHANNEL`** command selects which input channel waveform will be strobed.

The **`STROBECHANNEL`** query returns the presently selected strobe channel.

Command syntax- **`:SYST`**`em`**`:STRO`**`be`**`CHAN`**`nel<n>`

Example: `Send(0,5,”:SYSTem:STRObeCHANnel1",22,EOI);`

Query syntax- `:`**`:SYST`**`em`**`:STRO`**`be`**`CHAN`**`nel<n>`**`?`**

Example: `Send(0,5,”:SYSTem:STRObeCHANnel?”,22,EOI);`

5-22 STROBEDELAY

The **`STRObeDELay`** command is used to allow strobed voltage measurements along the pulses of a selected channel. Strobing is armed from External Arm.

With the same signal on a selected channel and on the selected arm channel, the strobed voltage value read will be 20ns from the beginning of the signal.

NOTE: To strobe at the beginning of a signal, delay the signal 20ns.

The **`STRObeDELay`** query returns the present strobe delay setting.

The range of delay settings is from 20,000ps to 100,000,000ps.

Command syntax- **`:SYST`**`em`**`:STRO`**`be`**`DEL`**`ay<value>(@<n,m,x,…>`**`|`**`<n:m>)`

Example: `Send(0,5,”:SYSTem:STRObeDELay20000",14,EOI);`

Query syntax- **`:SYST`**`em`**`:STRO`**`be`**`DEL`**`ay(@<n,m,x,…>`**`|`**`<n:m>)`**`?`**

Example: `Send(0,5,”:SYSTem:STRObeDELay?”,13,EOI);`

5-23 STROBEINCREMENT

The **STROBEINCREMENT** command sets the increment between strobe points. The increment is set in picoseconds.

The **STROBEINCREMENT** query returns the present strobe delay increment.

NOTE: For any given delay, resolution at that delay is better than 0.2% of the delay.

Command syntax- `:SYSTem:STRObeINCrement<value>(@<n,m,x,…>|<n:m>)`

Example: `Send(0,5,”:SYSTemSTRObeINCrement10000”,27,EOI);`

Query syntax- `:SYSTem:STRObeINCrement(@<n,m,x,…>|<n:m>)?`

Example: `Send(0,5,”:SYSTem:STRObeINCrement?”,24,EOI);`

5-24 STROBESTART

The **STROBESTART** command sets the start delay for the measure window command. The delay can be from 20,000ps to 100,000,000ps.

The **STROBESTART** query returns the present window start delay.

Command syntax- `:SYSTem:STRObeSTARt<value>(@<n,m,x,…>|<n:m>)`

Example: `Send(0,5,”SYSTem:STRObeSTARt20000”,25,EOI);`

Query syntax- `:SYSTem:STRObeSTARt(@<n,m,x,…>|<n:m>)?`

Example: `Send(0,5,”:SYSTem:STRObeSTARt?”,20,EOI);`

5-25 STROBESTEP

The **STRObeSTEPs** command sets the number of voltage measurement steps. The first measurement will be at the start value. The **STRObeSTEPs** query will return the present window number of steps value.

Command syntax- `:SYSTem:STRObeSTEPs<value>(@<n,m,x,…>|<n:m>)`

Example: `Send(0,5,”:SYSTem:STRObeSTEPs20”,17,EOI);`

Query syntax- `:SYSTem:STRObeSTEPs(@<n,m,x,…>|<n:m>)?`

Example: `Send(0,5,”:SYSTem:STRObeSTEPs?”,16,EOI);`
Response: <ASCII integer>

5-26 STROBESTOP

The **STROBESTOP** command sets the stop delay for the measure window command. The delay can be from 20,000ps to 100,000,000ps.

The **STROBESTOP** query returns the present window start delay.

Command syntax- `:SYSTem:STRObeSTOP<value>(@<n,m,x,…>|<n:m>)`

Example: `Send(0,5,”:SYSTem:STRObeSTOP100000”,24,EOI);`

Query syntax- `:SYSTem:STRObeSTOP(@<n,m,x,…>|<n:m>)?`

Example: `Send(0,5,”:SYSTem:STRObeSTOP?,19,EOI);`

5-27 TIMEOUT

The **TIMEOUT** command sets the timeout value, in seconds, to wait before reporting “No Pulses Found”, during a measurement.

The default, which is set at power up, is 10 seconds (floating point value).

Command syntax- `:SYSTem:TIMeout<value>`

Example: `Send(0,5,”:SYSTem:TIMeout15”,17,EOI);`

Query syntax- `:SYSTem:TIMeout?`

Example: `Send(0,5,”:SYSTem:TIMe?”,13,EOI);`

5-28 WAVE

The **WAVE** command selects the mode of pulsefind. Use FLAT to locate the flatspot of a square wave and use PEAK to find the peaks of a sine waveform.

The **WAVE** query returns the presently selected mode.

Command syntax- `:SYSTem:WAVe<PEAK|FLAT|STRObe>`

Example: `Send(0,5,”:SYSTem:WAVePEAK”,16,EOI);`

Query syntax- `:SYSTem:WAVe?`

Example: `Send(0,5,”:SYSTem:WAVe?”,13,EOI);`

NOTE: Use the `:ACQuire:LEVel(@ <n,m,x,…>|<n:m>)` command to perform the pulsefind.

5-29 WINDOW

The **WINDOW** command is a macro command to allow the parameter setup for the measure window command. A window can be from 20ns to100µs given in picoseconds.

To describe a window three (3) parameters must be given, window start delay, window stop delay and either the measure point increment or the number of points to make a measurement.

To set the parameters and return an average voltage measurement of the window, see the acquire window command.

Command syntax-
```
:SYSTem:WINDow/start value/stop value/<step increment
          |Number of steps>(@<n,m,x,…>|<n:m>)
```

Example 1: `Send(0,5,”:SYSTem:WINDow/20000/100000/10000”,33,EOI);`
Example 2: `Send(0,5,”:SYSTem:WINDow/20000/100000/N10”,31,EOI);`

This page intentionally left blank.

SECTION 6 - ACQUIRE COMMANDS

6-1 DESCRIPTION OF ACQUIRE COMMANDS

The **ACQUIRE** commands are used to set parameters used during a measure command.

`:ACQuire:<command syntax>`

Acquire commands:	**A**djacent**CYC**le	**HIST**ogram
	ANALysis	**JITT**er
	COMPlete	**LEV**el
	COUNt	**LOCK**time
	DATacom	**OSC**illoscope
	DRCG	**RAND**om**DAT**a
	DUTY	**SET**s**COUN**t
	EYEHistogram	**STAT**istics
	FUNCtion	**TIM**e**DIG**itizer
	GROUP	**TIM**e**SER**ies

Acquire Macros:	AC Measure -	**ALL**
		FUNCtion
		MEASure
		RUN
	DC Measure -	**WIND**ow

6-2 ADJACENTCYCLE

The **ADJACENTCYCLE** command allows the user to send a measurement request to the SIA3000 in the format of the Wavecrest API Adjacent Cycle structure. The operator would modify values of the ACYC structure on their system (host computer), then append that structure to the end of an `:ACQuire:AdjacentCYCle` command. The structure must be appended to the command in this format:

:ACQ:ACYC# xy...ssssssss...

x = an ASCII digit representing the number of digits in y
y = a string of digits, of x length, which represents the number of bytes of information to be returned.
s=the actual structure in binary format.

Upon exectution of the command, the measurement values are returned in the same format.

Command syntax- `:ACQuire:AdjacentCYCle(@<n,m,x,…>|<n:m>) <acyc_header><acyc_structure>`

Example: `Send(0,5,”:ACQuire:AdjacentCYCle@   ”,12,EOI);`

Query Command- NONE

6-3 ANALYSIS:FUNCTION|JITTER|RANGE

The **`ANALYSISFUNCTION`** command selects 1 of 10 functions and takes a measurement for the number of counts. The returned values are the mean of the measure, standard deviation, minimum and maximum in binary for each event where event is defined as a measurement. The returned values are in picoseconds except for frequency which returns the values in kilohertz.

Command syntax- `:ACQuire:ANALysisFUNCtion</FUNC/LowStartCount/HighStartCount /StopCountDesignator/Increment/DataDes>(@<n,m,x,…>|<n:m>)`

Example: `Send(0,5,":ACQuire:ANALysisFUNCtion/PW+/1/1/100/=/10/4",44,EOI);`
Example: `Send(0,5,":ACQuire:ANALysisFUNCtion/PER/2/1/100/+/10/4",44,EOI);`

If StopCount Designator = "+", Returns Stop Event
= "=", Returns Start Event
If DataDes = 2
Returns: Mean and standard deviation in binary
If DataDes = 4
Returns: The mean, standard deviation, minimum and maximum in binary.

Default: DataDes = 4

The **`ANALYSISJITTER`** command selects 1 of 10 functions and takes a measurement for the number of counts. The returned values are jitter, standard deviation, minimum and maximum in binary for each event where event is defined as a measurement. The returned value is in picoseconds, except for frequency which returns the values in kilohertz.

Command syntax- `ACQuire:ANALysisJITTer</FUNC/CHAN/StartCount/LowStopCount /HighStopCount/Increment/DataDes>`

Example: `Send(0,5,":ACQuire:ANALysisJITTer/PW+/1/1/1/100/10/3",42,EOI);`
Example: `Send(0,5,":ACQuire:ANALysisJITTer/PER/2/1/2/100/10/3",42,EOI);`

If DataDes = 3
Returns: Jitter, i.e., standard deviation, min, max in binary.
If DataDes = 2
Returns: Jitter, i.e., standard deviation and mean.

Default: DataDes = 3

The **`ANALYSISRANGE`** command is similar to the **`ANALYSISJITTER`** command except the returned value is the range, (Max –Min)/2, with minimum and maximum in binary for each event where event is defined as a measurement.

Command syntax- `:ACQuire:ANALysisRANGe</FUNC/CHAN/StartCount/LowStopCount /HighStopCount/Increment/DataDes>`

Example: `Send(0,5,":ACQuire:ANALysisRANGe/PW+/1/1/1/100/10/3",41,EOI);`
Example: `Send(0,5,":ACQuire:ANALysisRANGe/PER/2/1/2/100/10/3",41,EOI);`

If DataDes = 3
Returns: Range, min, max in binary.
If DataDes = 2
Returns: Range, standard deviation and mean.
Default: DataDes = 3

6-4 COMPLETE

The **`COMPLETE`** query returns the number of measurements completed for the specified channels. The returned value will be an ASCII integer value.

Command syntax- NONE

Query syntax- **`:ACQ`**`uire:`**`COMP`**`lete(@<n,m,x,…>|<n:m>)`**`?`**

Example: `Send(0,5,”:ACQuire:COMPlete?”,18,EOI);`
`Receive(0,5,data,1,EOI);`

Response: <ASCII count>

6-5 COUNT

The **`COUNT`** command sets the number of measurements used to develop the statistics, average, minimum, maximum, range and standard deviation for the specified channels. The number of measurements can range from 1 to 1,000,000. The **`COUNT`** query returns the present setting of the count value.

Command syntax- **`:ACQ`**`uire:`**`COUN`**`t<ASCII integer value>(@<n,m,x,…>|<n:m>)`

Example: `Send(0,5,”:ACQuire:COUNt200",17,EOI);`

Query syntax- **`:ACQ`**`uire:`**`COUN`**`t(@<n,m,x,…>|<n:m>)`**`?`**

Example: `Send(0,5":ACQuire:COUNt?”,15,EOI);`
`Receive(0,5,data,1,EOI);`

Response: <ASCII integer>

6-6 DATACOM

The **`DATacom`** command allows the user to send a measurement request to the `SIA3000` in the format of the Wavecrest API dataCOM structure. The operator would modify values of the DCOM structure on their system (host computer), then append that structure to the end of an `:ACQuire:DATacom` command. The structure must be appended to the command in this format:

`:ACQ:DAT#` `xy...ssssssss...`

x = an ASCII digit representing the number of digits in y
y = a string of digits, of x length, which represents the number of bytes of information to be returned.
s=the actual structure in binary format.

Upon exectution of the command, the measurement values are returned in the same format.

Command syntax- **`:ACQ`**`uire:`**`DAT`**`acom(@ <n,m,x,…>|<n:m>)<dcom_header>`
`<dcom_structure>`

Example: `Send(0,5,”:ACQuire:DATacom@`

Query syntax- NONE

6-7 DRCG

The **`DRCG`** command allows the user to send a measurement request to the `SIA3000` in the format of the Wavecrest API DRCG structure. The operator would modify values of the DRCG structure on their system (host computer), then append that structure to the end of an `:ACQuire:DRDG` command. The structure must be appended to the command in this format:

```
:ACQ:DRCG# xy...ssssssss...
```

x = an ASCII digit representing the number of digits in y

y = a string of digits, of x length, which represents the number of bytes of information to be returned.

s=the actual structure in binary format.

Upon exectution of the command, the measurement values are returned in the same format.

Command syntax- `:ACQuire:DRCG(@ <n,m,x,…>|<n:m>)<drcg_header>` `<drcg_structure>`

Example: `Send(0,5,”:ACQuire:DRCG@`

Query syntax- NONE

6-8 DUTY

The **`DUTY`** command will calculate the duty cycle of the signal and return a three digit ASCII number. The percent will be of the positive pulse width in a format of xx.x%.

Command syntax- `:ACQuire:DUTY(@ <n,m,x,…>|<n:m>)`

Example: `Send(0,5,”:ACQuire:DUTY”,12,EOI);`

Response: `49.8` (49.8%)

6-9 EYEHISTOGRAM

The **`EYEHISTOGRAM`** command allows the user to send a measurement request to the `SIA3000` in the format of the Wavecrest API EYEH structure. The operator would modify values of the EYEH structure on their system (host computer), then append that structure to the end of an `:ACQuire :EYEHISTogram` command. The structure must be appended to the command in this format:

```
:ACQ:EYEH# xy...ssssssss...
```

x = an ASCII digit representing the number of digits in y

y = a string of digits, of x length, which represents the number of bytes of information to be returned.

s=the actual structure in binary format.

Upon exectution of the command, the measurement values are returned in the same format.

Command syntax- `:ACQuire:EYEHistogram(@ <n,m,x,…>|<n:m>)` `<eyeh_header><eyeh_structure>`

Example: `Send(0,5,”:ACQuire:EYEHistogram@`

Query syntax- NONE

6-10 GROUP

After a user has defined a group (see Section 5-13 `:SYSTem:GROUP<ON|OFF>`), this command is called to execute all the commands that had been queued up in that particular group.

Command syntax- `:ACQuire:GROUP<1-20>`

Example: `Send(0,5,”:ACQuire:GROUP5”, 17, EOI);`

Query syntax- NONE

6-11 HISTOGRAM

The **`HISTOGRAM`** command allows the user to send a measurement request to the `SIA3000` in the formatof the Wavecrest API HIST structure. The operator would modify values of the HIST structure on their system (host computer), then append that structure to the end of an `:ACQuire:HISTogram` command. The structure must be appended to the command in this format:

```
:ACQ:HIST# xy...ssssssss...
```

x = an ASCII digit representing the number of digits in y
y = a string of digits, of x length, which represents the number of bytes of information to be returned.
s=the actual structure in binary format.

Upon exectution of the command, the measurement values are returned in the same format.

Command syntax- `:ACQuire:HISTogram(@ <n,m,x,…>|<n:m>)<hist_header> <hist_structure>`

Example: `Send(0,5,”:ACQuire:HISTogram@`

Query syntax- NONE

6-12 JITTER

The **`JITTER`** command allows the user to send a measurement request to the `SIA3000` in the format of the Wavecrest API JITT structure. The operator would modify values of the JITT structure on their system (host computer), then append that structure to the end of an `:ACQuire:JITTer`command. The structure must be appended to the command in this format:

```
:ACQ:JITT# xy...ssssssss...
```

x = an ASCII digit representing the number of digits in y
y = a string of digits, of x length, which represents the number of bytes of information to be returned.
s=the actual structure in binary format.

Upon exectution of the command, the measurement values are returned in the same format.

Command syntax- `:ACQuire:JITTer(@ <n,m,x,…>|<n:m>)<jitt_header> <jitt_structure>`

Example: `Send(0,5,”:ACQuire:JITTer@`

Query syntax- NONE

6-13 LOCKTIME

The **`LOCKTIME`** command allows the user to send a measurement request to the `SIA3000` in the format of the Wavecrest API LOCK structure. The operator would modify values of the LOCK structure on their system (host computer), then append that structure to the end of an `ACQuire:LOCKtime` command. The structure must be appended to the command in this format:

`:ACQ:LOCK#` `xy...ssssssss...`

x = an ASCII digit representing the number of digits in y

y = a string of digits, of x length, which represents the number of bytes of information to be returned.

s=the actual structure in binary format.

Upon exectution of the command, the measurement values are returned in the same format.

Command syntax- **`:ACQ`**`uire:`**`LOCK`**`time(@ <n,m,x,…>|<n:m>)<lock_header>`
`<lock_structure>`

Example: `Send(0,5,”:ACQuire:LOCKtime@`

Query syntax- NONE

6-14 LEVEL

The **`LEVEL`** command causes the instrument to find the pulse levels on the start and/or stop channels depending on the channel selection. If the arming source selected is external, the levels of the arming channels are found as selected.

The levels are stored and can later be read by using the channel commands. The percent of the peak level found will be displayed and returned as the new start and stop references.

The levels found for each channel are the minimum and maximum peak and the selected percentage of these peaks.

Command syntax- **`:ACQ`**`uire:`**`LEV`**`el(@ <n,m,x,…>|<n:m>)`

Example: `Send(0,5,”:ACQuire:LEVel@”,14,EOI);`

Query syntax- **`:ACQ`**`uire:`**`LEV`**`el(@ <n,m,x,…>|<n:m>)`**`?`**

6-15 RANDOMDATA

The **`RANDOMDATA`** command allows the user to send a measurement request to the `SIA3000` in the format of the Wavecrest API RAND structure. The operator would modify values of the RAND structure on their system (host computer), then append that structure to the end of an `ACQuire:RANDomDATa` command. The structure must be appended to the command in this format:

`:ACQ:LOCK#` `xy...ssssssss...`

x = an ASCII digit representing the number of digits in y

y = a string of digits, of x length, which represents the number of bytes of information to be returned.

s=the actual structure in binary format.

Upon exectution of the command, the measurement values are returned in the same format.

Command syntax- **`:ACQ`**`uire:`**`RAND`**`omDATa(@ <n,m,x,…>|<n:m>)`
`<rand_header><rand_structure>`

Example: `Send(0,5,”:ACQuire:RANDomDATa@`

Query syntax- NONE

6-16 SETSCOUNT

The **`SETSCOUNT`** command sets the count of a set of measurements which will create an average. This average is used with other set averages of sample size, to create the statistics available for return over the GPIB interface. The sets size value can range from 1 to 9999.

As an example, a sets size of a 100 and sample size of 1000 means that the statistics are of 10000 measurements of size 100.

The **`SETSCOUNT`** query returns the present setting of the sets size.

Command syntax- `:ACQuire:SETsCOUNt<ASCII integer value>`

Example: `Send(0,5,”:ACQuire:SETsCOUNt100",21,EOI);`

Query syntax- `:ACQuire:SETsCOUNt?`

Example: `Send(0,5,”:ACQuire:SETsCOUNt?”,19,EOI);`
Response: <ASCII setscount>

6-17 STATISTICS

The **`STATISTICS`** command allows the user to send a measurement request to the `SIA3000` in the format of the Wavecrest API STAT structure. The operator would modify values of the STAT structure on their system (host computer), then append that structure to the end of an `ACQuire:STATistics` command.

Command syntax-
```
:ACQuire:STATistics(@ <n,m,x,…>|<n:m>)
                      <stat_header><stat_structure>
```

Example: `Send(0,5,”:ACQuire:STATistics@`

Query syntax- NONE

6-18 TIMEDIGITIZER

The **`TIMEDIGITIZER`** command allows the user to send a measurement request to the `SIA3000` in the format of the Wavecrest API TDIG structure. The operator would modify values of the TDIG structure on their system (host computer), then append that structure to the end of an `ACQuire:TIMeDIGitizer` command.

Command syntax-
```
:ACQuire:TIMeDIGitizer(@ <n,m,x,…>|<n:m>)
                      <tdig_header><tdig_structure>
```

Example: `Send(0,5,”:ACQuire:TIMeDIGitizer@`

Query syntax- NONE

6-19 TIMESERIES

The **`TIMESERIES`** command allows the user to send a measurement request to the `SIA3000` in the format of the Wavecrest API TSER structure. The operator would modify values of the TSER structure on their system (host computer), then append that structure to the end of an `ACQuire:TimeSERies` command.

Command syntax-
```
:ACQuire:TIMeSERies(@ <n,m,x,…>|<n:m>)
                      <tser_header><tser_structure>
```

Example: `Send(0,5,”:ACQuire:TIMeSERies@`

Query syntax- NONE

ACQUIRE MACROS

6-20 ALL

The **`ALL`** command will select 1 of 11 functions, take a measurement and return the average, standard deviation, minimum and maximum. The function selected will force the following parameters to defaults:

Edges - Rising or falling

Channel - Single or both (if a single channel function, start or stop will be selected based on last single channel selected.

Arming - Auto-on-start, auto-on-stop, start first or stop first, based on the last arming sequence selected for that function.

Command syntax- `:ACQuire:ALL<TT+|TT-|PW+|PW-|PERiod+|PERiod-|TPD++|TPD- -|TPD+-|TPD-+|FREQ>`

Example:
```
Send(0,5,”:ACQuire:ALLTT+”,15,EOI);
Receive(0,5,data,4,EOI);
```

Query syntax- None

6-21 FUNCTION

The **`FUNCTION`** command will select 1 of 11 functions that will guide the instrument during time measurements. The function selected will force the follow parameters to defaults:

Edges - Rising or falling

Channel - Single or both (if a single channel function, start or stop will be selected based on last single channel selected.

Arming - Auto-on-start, auto-on-stop, start first or stop first, based on the last arming sequence selected for that function.

The **`FUNCTION`** query will return the presently selected function.

Command syntax- `:ACQuire:FUNCtion<TT+|TT-|PW+|PW-|PERiod+|PERiod-|TPD++|TPD- -|TPD+-|TPD-+|FREQ>`

Example: `Send(0,5,”ACQuire:FUNCtionTT+”,19,EOI);`

Query syntax- `:ACQuire:FUNCtion?`

Example: `Send(0,5,”:ACQuire:FUNCtion?”,18,EOI);`
Response: `<TT+|TT-|PW+|PW-|PER|TPD++|TPD—|TPD+-|TPD-+|FREQ>`

6-22 MEASURE

The **`MEASURE`** command will take a time measurement and return the average and standard deviation. The present function and reference voltages are used. This is a fast method of performing the acquire run command repetitively.

Command syntax- `:ACQuire:MEASure`

Example:
```
Send(0,5,”:ACQuire:MEASure”,16,EOI);
Receive(0,5,data,2,EOI);
```

Query syntax- None

6-23 RUN

The **RUN** command will select 1 of 10 functions, take a measurement and return the average and standard deviation. The function selected will force the following parameters to defaults:

Edges **-** Rising or falling

Channel **-** Single or both (if a single channel function, start or stop will be selected based on last single channel selected.

Arming **-** Auto-on-start, auto-on-stop, start first or stop first, based on the last arming sequence selected for that function.

Command syntax- **:ACQ**uire**:RUN**<TT+|TT-|PW+|PW-|PERiod+|PERiod- |TPD++|TPD- -|TPD+-|TPD- +|FREQ>(@<n,m,x,…>|<n:m>)

Example:
```
Send(0,5,”:ACQuire:RUNTT+”,15,EOI);
Receive(0,5,data,1,EOI);
```

Query syntax- None

6-23 WINDOW

The **WINDOW** command is a macro to set parameters and return the average (mean) voltage of the window. The window can be of a delay from 20,000ps to 100,000,000ps.

To describe a window, three (3) parameters can be given. If any parameter is omitted the forward slash (/) must be placed in the command to indicate the proper spacing.

The three parameters are:

start delay value	20,000ps to 100,000,000ps
stop delay value	20,000ps to 100,000,000ps
increment between points	see system strobe increment command
or	
number of measurement points	see system strobe points command

Command syntax- **:ACQ**uire**:WIND**ow/start value/stop value/<step increment|#of points>

Example 1:
```
Send(0,5,”:ACQuire:WINDow/25000/50000/1000”,32,EOI);
Receive(0,5,voltage level,5,EOI);
```
Example 2:
```
Send(0,5,”:ACQuire:WINDow/25000/50000/#100”,32,EOI);
Receive(0,5,voltage level,5,EOI);
```

Query syntax- None

This page intentionally left blank.

SECTION 7 - CALIBRATE COMMANDS

7-1 DESCRIPTION OF CALIBRATE COMMANDS

The **CALIBRATE** commands enables the host to perform an internal or external calibration and set or read the external calibration values.

```
:CALibrate:<command syntax>
```

Calibrate commands:	**DATA**	**INT**ernal
	DESKEW	**STAT**us
	DESKEWDC	e**X**tented **INT**ernal

7-2 DATA

The **DATA** command can be used to enable the host to write the individual channel skew values to the instrument.

There are 10 skew values (one value per possible SIA channel) that must be sent to the device from the host in the following format (ANSI/IEEE Std. 754-1985 floating-point standard):

#xy..dddddddddd.., where:
x = an ASCII digit representing the number of digits in y
y = a string of digits, of x length, which represents the number of bytes of information to be sent.
d = calibration data

The DATA query command is used to read the 10 values that are returned in the same floating-point format (ANSI/IEEE Std. 754-1985).

Command Syntax- **:CAL**ibrate**:DATA**<#xy..dddddddddd..>

Example: `Send(0,5, ":CALibrate:DATA#280`<80 bytes of data (10 skew values - 10x8)>`", 99, EOI);`

Query Syntax- **:CAL**ibrate**:DATA?**

Example: `Send(0,5, ":CALibrate:DATA?", 20, EOI);`
Response: #280<80 bytes of data(10 skew values - 10x8)>

7-3 DESKEW

The **DESKEW** command permits the user to perform the same Deskew calibration remotely over GPIB rather than on the SIA3000 front panel (*VISI*™).

Command syntax- :CALibrate**:DESKEW**

Example: `Send(0,5,":CALibrate:DESKEW",17,EOI);`

Query syntax- None

7-4 DESKEWDC

The **`DESKEWDC`** command permits the user to perform the same Deskew with DC calibration remotely over GPIB rather than using the SIA3000 front panel (*VISI*).

Command syntax- **`:CAL`**`ibrate:`**`DESKEWDC`**

Example: `Send(0,5,”:CALibrate:DESKEWDC”,19,EOI);`

Query syntax- None

7-5 INTERNAL

The **`INTERNAL`** command permits the user to perform the same internal (Timer) calibration remotely over GPIB rather than using the SIA3000 front panel (*VISI*).

Command syntax- **`:CAL`**`ibrate:`**`INT`**`ernal`

Example: `Send(0,5,”:CALibrate:INTernal”,19,EOI);`

Query syntax- None

See Appendix A for a more complete example.

7-6 INTERNAL CALIBRATION - EXTENDED

The internal calibration function will process 20,000,000 samples while taking 11 minutes to complete. With e**X**tended **INT**ernal calibration, the user can set a multiplier, 5.5 minutes, of the sample count from 1 to 25. Time for completion is extended by the same multiplier factor. A setting of 6 is recommended.

Extended Internal Calibration allows the user to possibly reduce jitter due to the noise floor of the instrument through the use of longer internal calibration periods. The selected multiplier, from 1 to 25, extends the base calibration period of approximately 5.5 minutes by that factor.

Command Syntax- **`:CALibrate:XINTernal`**`<ASCII VALUE>`

Example: `Send(0,5,“:Calibrate:XINTernal6”,21,EOI”);`

Query Syntax- **`:CAL`**`ibrate:`**`XINT`**`ernal`**`?`**

Example: `Send(0,5,”:CALibrate:XINTernal?”,21,EOI);`

Response: 6

7-7 STATUS

The **`STATus`** query will return the current status of the calibration tests. (0 = pass, 1 = fail):

Command Syntax- None

Query syntax- **`:CAL`**`ibrate:`**`REF?`**

Example: `Send(0,5,”:CALibrate:REF?”,15,EOI);`

Bit Pos.	Hex Value	Description
0	1	Internal Calibration
1	2	Deskew Calibration
3	4	DeskewDC Calibration
4	8	Strobe Calibration

SECTION 8 - CHANNEL COMMANDS

8-1 DESCRIPTION OF CHANNEL COMMANDS

The **`CHANNEL`** commands write and read the channel start and stop reference voltages, the arm on *n*th counts and external arming selections..

`:CHAN``nel<n><`**`STAR`**`t|`**`STOP`**`>:<command syntax>`

Channel commands:	**`COUN`**`t`	**`LEV`**`el`	**`SWIT`**`ch`
	`EXT``ernalarm`	**`MIN`**`imum/`**`MAX`**`imum`	

8-2 COUNT

The **`COUNT`** command sets the arm-on-*n*th-event count for either the start or stop event. The range of
the nth event is from 1 to 131072.

The **`COUNT`** query returns the count of either the start or stop event.

Command syntax- **`:CHAN`**`nel<n><`**`STAR`**`t|`**`STOP`**`>:`**`COUN`**`t<value>`

Example: `Send(0,5,”:CHANnel4STARt:COUNt100",23,EOI);`

Query syntax- **`:CHAN`**`nel<n><`**`STAR`**`t|`**`STOP`**`>:`**`COUN`**`t`**`?`**

Example: `Send(0,5,”:CHANnel4STOP:COUNt?”,20,EOI);`
Response: <ASCII count>

8-3 EXTERNALARM

The **`EXTERNALARM`** command selects which arming channel is associated with the start and stop events.

The **`EXTERNALARM`** query returns the arming selected for a specific (start/stop) event.

Command syntax- **`:CHAN`**`nel<n>:`**`EXT`**`ernalarm<a>`

Example: `Send(0,5,”:CHANnel4:EXTernalarmARM1",25,EOI);`

Query syntax- **`:CHAN`**`nel<n>:`**`EXT`**`ernalarm`**`?`**

Example: `Send(0,5,”:CHANnel4:EXTernalarm?”,22,EOI);`
Response: <`ARM1|ARM2`>

8-4 LEVEL

The **`LEVEL`** command sets the start/stop reference levels. The range is ±1.1 volts in 150 microvolt resolution.

The **`LEVEL`** query returns the start/stop levels. The level returned is an integer value.

Command syntax- **`:CHAN`**`nel<n><`**`STAR`**`t|`**`STOP`**`>:`**`LEV`**`el<value>`

Example: `Send(0,5,”:CHANnel4STARt:LEVel+1.33",25,EOI);`

Query syntax- **`:CHAN`**`nel<n>`**`STAR`**`t:`**`LEV`**`el`**`?`**

Example: `Send(0,5,”:CHANnel4STARt:LEVel?”,21,EOI);`
Response: <value>
Example: `+1.10000`

See Appendix C for more information regarding returned data formats.

8-5 MINIMUM/MAXIMUM

The **`MINIMUM/MAXIMUM`** query returns the minimum or maximum peak levels of the start or stop reference levels. The peak values measured when the last pulse find was initiated. This pulsefind could have been initiated from the front panel or with the `:Acquire:Level` command.

Command syntax: None

Query syntax- `:CHANnel<n><STARt|STOP>:<MIN|MAX>?`

Example: `Send(0,5,”:CHANnel4STARt:MIN?”,19,EOI);`

Response: <ASCII MIN or MAX peak level>

Example: `-1.01890`

8-6 SWITCH ON/OFF

The **`SWITCH ON/OFF`** command enables or disables the switches on the front panel of the DSM-16.

Command syntax- `:CHANnel:SWITch<ON|OFF>`

Example: `Send(0,5,”:CHANnel:SWITchON”,17,EOI);`

Query syntax- None

8-7 SWITCH IDN

The **`SWITCH IDN`** query returns the version number of the DSM–16. The returned value is an ASCII number representing the version major and minor (i.e. 1.1).

Command syntax- None

Query syntax- `:CHANnel:SWITchIDN?`

Example: `Send(0,5,”:CHANnel:SWITchIDN?”,19,EOI);`
Response: <ASCII number `1-16`>

8-8 CHANNEL:SWITCH number

The **`SWITCH`** number command identifies the instrument’s input channel to be selected. The DSM–16 is designed to be used as a 1 of 8 matrix to the instrument’s channel (1 of 8 to channel X, and 1 of 8 to channel Y). The matrix inputs are assigned channel numbers 11-18 and 21-28.

NOTE: A small number of units are labeled 1 through 16.

Command syntax- `:CHANnel:SWITch<11...18|21...28>`

Example: `Send(0,5,”:CHANnel:SWITch15",17,EOI);`

This will select the left bank of eight, fifth input from the left.

NOTE: The DSM-16 can be configured as a 1 of 15 matrix by connecting the eighth input from the left bank to the Channel 2 output.

Query syntax- `:CHANnel:SWITch?`

Example: `Send(0,5,”:CHANnel:SWITch?”,16,EOI);`

Response: <ASCII digits>

The returned format is ASCII digits representing both channel/switch numbers separated by a space (i.e. 15 21).

Example: `15 21`

SECTION 9 - DISPLAY COMMANDS

9-1 DESCRIPTION OF DISPLAY COMMANDS

The **DISPLAY** commands control the displaying of information on the front panel and if filtering is used in developing the statistics.

```
:DISPlay:<command syntax>
```

Display commands:	**FILT**er –	ON/OFF
		MIN/MAX (Limits)
	LEVel	
	USER	

9-2 FILTER (ON/OFF)

The **FILTER** (ON/OFF) command is used to select whether filtering will be used in calculating the statistics.

The **FILTER** query returns the current ON/OFF selection.

Command syntax- `:DISPlay:FILTer<ON|OFF>`

Example: `Send(0,5,”:DISPlay:FILTerON”,17,EOI);`

Query syntax- `:DISPlay:FILTer?`

Example: `Send(0,5,”:DISPlay:FILTer?”,16,EOI);`
Response: `<ON|OFF>`

9-3 FILTER (Limits)

The **FILTER** (limits) query commands return the ASCII floating point value of the presently set limits. Limits are ±2.4999999999 seconds. Maximum setting must be greater than Minimum setting.

A value of -999,999,999 is returned by the measure deviation query command if there were not any measurements within the limits.

Command syntax- `:DISPlay:FILTer<MINimum|MAXimum>`
`<signed ASCII floating point>`

Example: `Send(0,5,”:DISPlay:FILTerMINimum+0.000000000500",37,EOI);`

Query syntax- `:DISPlay:FILTer<MINimum|MAXimum>?`

Example: `Send(0,5,”:DISPlay:FILTerMINimum?”,23,EOI);`
Response: <Signed ASCII floating point>
Example: `-5.000000e+001` (-0.5)

9-4 LEVEL

The **`LEVEL`** command sets the start and stop percentage level of peaks that the start and stop references will be set to. This percentage of peak will also be what the front panel start and stop references will be displaying.

The **`LEVEL`** query will return a start and stop percentage setting in two ASCII integers.

Command syntax- **`:DISP`**`lay`**`:LEV`**`el<value value>`

Example: `Send(0,5,”:DISPlay:LEVel20 80",19,EOI);`

Query syntax- **`:DISP`**`lay`**`:LEV`**`el`**`?`**

Example: `Send(0,5,”:DISPlay:LEVel?”,15,EOI);`
Response: <value value>
Example: `20 80`

9-5 USER

The **`USER`** command selects the user set of references of the current function.

The instrument is capable of having the reference voltages set by two (2) methods.

1. Doing a pulse find and setting the references to a percentage of the peaks found.
2. Setting the start and stop voltage trip reference to a value.

When the user set the reference voltages directly, this is defined as a USER setting and is later selected by the display user command.

The **`USER`** query returns the setting of the user reference voltages.

Command syntax- **`:DISP`**`lay`**`:USER`**`<ON|OFF>`

Example: `Send(0,5,”:DISPlay:USERON”,15,EOI);`

Query syntax- **`:DISP`**`lay`**`:USER?`**

Example: `Send(0,5,”:DISPlay:USER?”,14,EOI);`
Response: `<ON|OFF>`

SECTION 10 - MEASURE COMMANDS

10-1 DESCRIPTION OF MEASURE COMMANDS

The **MEASURE** query returns the measurement statistics from the instrument to a host.

```
:MEASure:<command syntax>
```

Measure commands:	Time Measurement	DC Measurement
	AVERage	**Single** -
	DATA (Float or Double)	**DC**level
	DATAT	**STRO**be**VLEV**el
	EVENt	**Multiple** -
	JITTer	**VDATA**
	MAXimum	**VDATA4**
	MINimum	**VMAX**imum
	RANGe	**VMIN**imum
	Standard **DEV**iation	**VSDEV**iation
	STAT4	**WIND**ow

10-2 AVERAGE

The **AVERAGE** command returns the measured average of 1 to 1,000,000 measurements. The returned value is an ASCII floating point number.

Command syntax- None

Query syntax- **:MEAS**ure**:AVER**age(@ <n,m,x,…>|<n:m>)**?**

Example: `Send(0,5,":MEASure:AVERage?",17,EOI);`
Response: <ASCII floating point>
Example: `-8.4566284e-011`

10-3 DATA/DATA4

The **DATA** query returns a selected number of measured values. These measured data values can be analyzed or used to provide a presentation. The Measure Data query supports two sizes of data types (See Appendix C) using IEEE standards for floating-point arithmetic (ANSI/IEEE Std. 754-1985). The returned data stream is of the following format:

:MEASure**:DATAT** xy...dddddddd...

x = an ASCII digit representing the number of digits in y
y = a string of digits, of x length, which represents the number of bytes of information to be returned.
d=data

Command syntax- None

Query syntax-

Float — **:MEAS**ure**:DATA4**(@ <n,m,x,…>|<n:m>)**?**

`:MEASure:Data#43200`<200 bytes of data (50 measurements – 50x4)>

Double — **:MEAS**ure**:DATA**(@ <n,m,x,…>|<n:m>)**?**

`:MEASure:Data#3400`<400 bytes of data (50 measurements – 50x8)>

Example: `char data[2048]`
```
Send(0,5,:MEASure:DATA4?,15,EOI);
Receive(0,5,data,205,EOI);
```
Example: `char data[2048]`
```
Send(0,5,:MEASure:DATA?,14,EOI);
Receive(0,5,data,405,EOI);
```

10-4 DATAT

The **DATAT** query command returns the elapsed time measurements from a previous burst after the elapsed time counter has been turned on. With a sample size of 100 there will be 100 floating point time measurements returned.

The **DATAT** query returns a selected number of measured values. These measured data values can be analyzed or used to provide a presentation. See Appendix C for the returned data stream format types:

:MEASure:DATAT `xy...dddddddd...`

x = an ASCII digit representing the number of digits in y

y = a string of digits, of x length, which represents the number of bytes of information to be returned.

d=data

Query syntax- **:MEAS**ure**:DATAT**`(@ <n,m,x,…>|<n:m>)`**?**

Float — **:MEAS**ure**:DATAT?**

Example: `char data[2048]`
```
Send(0,5,”:MEASure:DATAT?”,15,EOI);
Receive(0,5,data,205,EOI);
```

10-5 JITTER

The **JITTER** query returns the standard deviation of the selected sample size.

Command syntax- None

Query syntax- **:MEASure:JITT**er`(@ <n,m,x,…>|<n:m>)`**?**

Example: `Send(0,5,”:MEASure:JITTer?”,16,EOI);`
Response: <ASCII floating point>
Example: `+7.3441603e-012`

10-6 MAX

The **MAX** query command returns the maximum measured value of a set of measurements.

Command syntax- None

Query syntax- **:MEAS**ure**:MAX**`(@ <n,m,x,…>|<n:m>)`**?**

Example: `Send(0,5,”:MEASure:MAX?”,13,EOI);`
Response: <ASCII floating point>
Example: `-6.5307617e-011`

10-7 MIN

The **MIN** query command returns the minimum measured value of a set of measurements.

Command syntax- None

Query syntax- `:MEASure:MIN(@ <n,m,x,…>|<n:m>)?`

Example: `Send(0,5,”:MEASure:MIN?”,13,EOI);`
Response: <ASCII floating point>
Example: `-1.1169434e-010`

10-8 RANGE

The **RANGE** query command returns the plus or minus difference between the maximum and minimum values of a set of measurements.

Command syntax- None

Query syntax- `:MEASure:RANGe(@ <n,m,x,…>|<n:m>)?`

Example: `Send(0,5,”:MEASure:RANGe?”,15,EOI);`
Response: <ASCII floating point>

10-9 SDEVIATION

The **SDEVIATION** query returns the standard deviation of the selected sample size.

Command syntax- None

Query syntax- `:MEASure:SDEViation(@ <n,m,x,…>|<n:m>)?`

Example: `Send(0,5,”:MEASure:SDEViation?”,20,EOI);`
Response: <ASCII floating point>
Example: `+7.3441603e-012`

10-10 STAT(istics)4

The **STAT4** query returns statistical data defined by `:SYST:STAT` for multiple SETS of measurements as float. The `:SYST:STAT/ON` command must be executed prior to using the **STAT4** command. Statistics are always returned in the order of AV, JI, MN and MX, depending on which ones are selected.

Command syntax- `:MEASure:STAT4(@ <n,m,x,…>|<n:m>)?`

Example: `Send(0,5,”:MEASure:STAT4?”,15,EOI);`
Response: <4-Byte float (Intel)>
Example: <`ON|OFF`><AV><JI><MN><MX>

Query syntax- None

DC MEASUREMENT - Single

10-11 DCVLEVEL

The **`DCVLEVEL`** command returns the dc voltage measured on the selected input channel. The returned value is an ASCII string of five digits preceded by a (+) or (-) sign. The value is a signed integer with 100 microvolt resolution.

Command syntax- `:MEASure:DCvlevel(@ <n,m,x,…>|<n:m>)?`

Example: `Send(0,5,”:MEASure:DCvlevel?”,18,EOI);`
Response: `-1.1444092e-004`

Query syntax- None

10-12 STROBEVLEVEL

The **`STROBEVLEVEL`** query returns the strobed dc voltage measured on the input channel selected. The strobing is provided through the arming channel. The strobing arm point can be be controlled by the strobe delay or by external moving the arming signal.

The returned value is an ASCII string of five (5) digits preceded by a (+) or (-) sign. The value is a signed integer with 100 microvolt resolution.

To perform a strobed measurement, set up the following parameters:

`STRObe CHANnel`

`STRObe ARMing`

`STRObe DELay`

Command syntax- None

Query syntax- `:MEASure:STRObeVLEVel(@ <n,m,x,…>|<n:m>)?`

Example:
```
Send(0,5,”:SYSTem:STRObeCHANnel1”,22,EOI);
Send(0,5,”:SYSTem:STRObeARMARM1”,21,EOI);
Send(0,5,”:SYSTem:STRObeDELay25000”,24,EOI);
Send(0,5,”:MEASure:STRObeVLEVel?”,21,EOI);
Receive(0,5,voltage level,5,EOI);
```
Response: <ASCII floating point>
Example: `-2.1731481e-003`

To perform multiple measurements that are averaged, see the **`:MEASure:WINDow`** command.

DC MEASUREMENT - Multiple

10-13 VDATA

The **VDATA** query returns the voltage measurement points acquired in the previous measure window or acquire window command. The measured data values can be analyzed or used to provide a presentation.

Each voltage value is returned in 5 digits preceded by a (+) or (-) sign. The returned voltage is an ASCII integer string of 100 microvolt resolution.

Example: `+0.0001` (+100 uv) would be +1
`-1.0` (-1 v) would be -1000

The returned data stream is of the following format:

:MEASure**:VDATA** xy...dddddddd...

x = an ASCII digit representing the number of digits in y

y = an ASCII string of digits, of x length, which represents the number of bytes of information to be returned.

d=data

Command syntax- None

Query syntax- **:MEAS**ure:**VDATA**(@ <n,m,x,…>|<n:m>)**?**

Example:
```
char data [2048];
Send(0,5,”:MEASure:VDATA?”,15,EOI);
Receive(0,5,data,60,EOI);
```

10-14 VDATA4

The **VDATA4** query is the same as the VDATA command except that the data is returned as float for throughput. (See VDATA, Section 10-14.) See Appendix C for returned formats.

Command syntax- None

Query syntax- **:MEAS**ure:**VDATA4**(@ <n,m,x,…>|<n:m>)**?**

Example: `Send(0,5,”:MEASure:VDATA4?”,16,EOI);`

10-15 VMAXIMUM

The **VMAXIMUM** query returns the maximum voltage value measured in the previous measure window or acquire window command.

Command syntax- None

Query syntax- **:MEAS**ure:**VMAX**imum(@ <n,m,x,…>|<n:m>)**?**

Example:
```
Send(0,5,”:MEASure:VMAXimum?”,18,EOI);
Receive(0,5,voltage level,5, EOI);
```
Response: <Signed ASCII value>
Example: `-0.00758`

10-16 VMINIMUM

The **`VMINIMUM`** query returns the minimum voltage value measured in the previous measure window or acquire window command.

Command syntax- None

Query syntax- `:MEASure:VMINimum(@ <n,m,x,…>|<n:m>)?`

Example: `Send(0,5,”:MEASure:VMINimum?”,18,EOI);`
`Receive(0,5,voltage level,5,EOI);`
Response: <Signed ASCII value>
Example: `-0.00821`

10-17 VSDEVIATION

The **`VSDEVIATION`** query returns the voltage standard deviation of the previous measure window or acquire window command. The returned value is a 6-digit ASCII string of a decimal number.

Command syntax- None

Query syntax- `:MEASure:VSDEViation(@ <n,m,x,…>|<n:m>)?`

Example:
```
char data [10];
Send(0,5,”:MEASure:VSDEViation?”,21,EOI);
Receive(0,5,data,7,EOI);
```
Response: <Signed ASCII value>

10-18 WINDOW

The **`WINDOW`** query instructs the instrument to take a series of strobed voltage measurements and then returns the average (mean) voltage. The following statistics are also available upon completion of the command.

VMAXIMUM	Maximum voltage measured
VMINIMUM	Minimum voltage measured
VSDEVIATION	Standard deviation of voltages measured

The following parameters must be set up prior to sending a measure window query:

`STRObe CHANnel`	Select channel to be strobed
`STRObe ARMing channel`	Select strobing (arming) input
`STRObe STARting point delay`	Set delay for the first strobed point
`STRObe STOPping point delay`	Set delay for the last strobed point
`STRObe INCRement between points`	Set increment between strobed points. (the instrument will calculate the number of points between start and stop)
`MEASure WINdow?`	Takes measurements and returns average

NOTE: Strobe increment defines a delay between each measurement. The instrument determines how many points to measure. An alternate method is to define the number of points between the first and last delayed points (`:SYSTem:STRObe#`) and the instrument will determine the delay increment between measured points.1

Example:
```
Send(0,5,":SYSTem:STRObeCHANnel1",22,EOI);
Send(0,5,":SYSTem:STRObeARMARM1",21,EOI);
Send(0,5,":SYSTem:STRObeSTArt25000",24,EOI);
Send(0,5,":SYSTem:STRObeSTOP50000",23,EOI);
Send(0,5,":SYSTem:STRObeINCRememt1000",27,EOI);
Send(0,5,":MEASure:WINdow?",16,EOI);
Receive(0,5,voltage level,6,EOI);
```

To measure a single strobe point, see the STROBEVLEVEL command.

To use a macro type of command to set up delays, take the measurements and return the voltage average, see the `:ACQuire:WINdow` command.

Command syntax- None

Query syntax- `:MEASure:WINDow(@ <n,m,x,…>|<n:m>)?`

Example:
```
Send(0,5,":MEASure:WINDow?",16,EOI);
Receive(0,5,data,5,EOI);
```
Response: <Signed ASCII value>

Example: `-0.00758`

This page intentionally left blank.

SECTION 11 - PLOT COMMANDS

11-1 DESCRIPTION OF PLOT COMMANDS

The **PLOT** commands read measurement data (`:ACQuire:<API structure>`) that is stored in dynamically-allocated memory within the SIA-3000. In order to pass this data onto the user, the user must specifically request it.

```
:PLOT:<API structure>(@n)<plot offset value>
```

Plot commands:			
	Adjacent**CYC**le	**HIST**ogram	**RAND**om**DAT**a
	DATacom	**JITT**er	**TIM**e**DIG**itizer
	DRCG	**LOCK**time	**TIM**e**SER**ies
	EYEHistogram	**OSC**illoscope	

11-2 ADJACENTCYCLE

The **ADJACENTCYCLE** command gathers graphical (plot) data acquired from the latest `:ACQuire:AdjacentCYCle` command for the channel specified.

Command syntax- `:PLOT:AdjacentCYCle(@n)<ACYC plot offset value>`

Example: `Send(0,5,”:PLOT:AdjacentCYCle%d",21,EOI);`

Query syntax- None

11-3 DATACOM

The **DATACOM** command gathers graphical (plot) data acquired from the latest `:ACQuire :DATacom` command for the channel specified.

Command syntax- `:PLOT:DATacom(@n)<DAT plot offset value>`

Example: `Send(0,5,”:PLOT:DATacom%d",15,EOI);`

Query syntax- None

11-4 DRCG

The **DRCG** command gathers graphical (plot) data acquired from the latest `:ACQuire:DRCG` command for the channel specified.

Command syntax- `:PLOT:DRCG(@n)<DRCG plot offset value>`

Example: `Send(0,5,”:PLOT:DRCG%d",12,EOI);`

Query syntax- None

11-5 EYEHISTOGRAM

The **`EYEHISTOGRAM`** command gathers graphical (plot) data acquired from the latest `:ACQuire:EYEHistogram` command for the channel specified.

Command syntax- **`:PLOT:EYEH`**`istogram(@n)<EYEH plot offset value>`

Example: `Send(0,5,”:PLOT:EYEHistogram%d",20,EOI);`

Query syntax- None

11-6 HISTOGRAM

The **`HISTOGRAM`** command gathers graphical (plot) data acquired from the latest `:ACQuire:HISTogram` command for the channel specified.

Command syntax- **`:PLOT:HIST`**`ogram(@n)<HIST plot offset value>`

Example: `Send(0,5,”:PLOT:HISTogram%d",17,EOI);`

Query syntax- None

11-7 JITTER

The **`JITTER`** command gathers graphical (plot) data acquired from the latest `:ACQuire:JITTer` command for the channel specified.

Command syntax- **`:PLOT:JITT`**`er(@n)<JITT plot offset value>`

Example: `Send(0,5,”:PLOT:JITTer%d",14,EOI);`

Query syntax- None

11-8 LOCKTIME

The **`LOCKTIME`** command gathers graphical (plot) data acquired from the latest `:ACQuire:LOCKtime` command for the channel specified.

Command syntax- **`:PLOT:LOCK`**`time(@n)<FUNC plot offset value>`

Example: `Send(0,5,”:PLOT:LOCKtime%d",16,EOI);`

Query syntax- None

11-9 OSCILLOSCOPE

The **`OSCILLOSCOPE`** command gathers graphical (plot) data acquired from the latest `:ACQuire:OSCilloscope` command for the channel specified.

Command syntax- **`:PLOT:OSC`**`illoscope(@n)<OSC plot offset value>`

Example: `Send(0,5,”:PLOT:OSCilloscope%d",20,EOI);`

Query syntax- None

11-10 RANDOM DATA

The **`RANDOM DATA`** command gathers graphical (plot) data acquired from the latest `:ACQuire:RANDomDATa` command for the channel specified.

Command syntax- `:PLOT:RANDomDATa(@n)<RANDDAT plot offset value>`

Example: `Send(0,5,”:PLOT:RANDomDATa%d",25,EOI);`

Query syntax- None

11-11 TIMEDIGITIZER

The **`TIMEDIGITIZER`** command gathers graphical (plot) data acquired from the latest `:ACQuire:TIMeDIGitizer` command for the channel specified.

Command syntax- `:PLOT:TIMeDIGitizerme(@n)<TDIG plot offset value>`

Example: `Send(0,5,”:PLOT:TIMeDIGitizertime%d",25,EOI);`

Query syntax- None

11-12 TIMESERIES

The **`TIMESERIES`** command gathers graphical (plot) data acquired from the latest `:ACQuire:TIMeSERies` command for the channel specified.

Command syntax- `:PLOT:TIMeSERies(@n)<TSER plot offset value>`

Example: `Send(0,5,”:PLOT:TIMeSERies%d",18,EOI);`

Query syntax- None

This page intentionally left blank.

SECTION 12 - TRIGGER COMMANDS

12-1 DESCRIPTION OF TRIGGER COMMANDS

The **`TRIGGER`** commands control the source and the level of the arming signal.

`:`**`TRIG`**`ger:<command syntax>`

Trigger commands:	**`DEL`**`ay`	**`SEQ`**`uence (start/stop)`
	`LEV``el`	`Arm `**`SLOP`**`e (Edge)`
	`MIN``imum/`**`MAX`**`imum`	**`SOUR`**`ce (external/automatic)`

12-2 DELAY

The **`DELAY`** command gives the remote operator the ability to set the Arming Delay just like on the SIA3000 front panel. Instead of the user entering a time value between 19 to 21 ns, the user sends a positive or negative increment value from the nominal arming delay to achieve the same effect.

Command syntax- `:`**`TRIG`**`ger:`**`DEL`**`ay<step value>`

Example: `Send(0,5,”:TRIGger:DELay1 0.1",21,EOI);`

Query syntax- `:`**`TRIG`**`ger:`**`DEL`**`ay`**`?`**

Example: `Send(0,5,”:TRIGger:DELay2?”,19,EOI);`

12-3 LEVEL

The **`LEVEL`** command sets the trip level of the arming input. The levels that can be selected are ±1.11 volts.

The **`LEVEL`** query returns the present trip setting of the specific arming input. The value is a 5-digit ASCII floating point number.

Command syntax- `:`**`TRIG`**`ger:`**`LEV`**`el<value>`

Example: `Send(0,5,”:TRIGger:LEVel1 0.1",21,EOI);`

Query syntax- `:`**`TRIG`**`ger:`**`LEV`**`el<n>`**`?`**

Example: `Send(0,5,”:TRIGger:LEVel2?”,19,EOI);`
Response: `<value>`
Example: `+1.11000`

12-4 MINIMUM/MAXIMUM

The **`MINIMUM/MAXIMUM`** query returns the minimum or maximum peak levels of the ARM reference levels. The peak values were measured when the last pulse find was initiated. This pulse find could have been initiated with the `:`**`ACQ`**`uire:`**`LEV`**`el` command.

Command syntax- None

Query syntax- `:`**`TRIG`**`ger:<MAXimum|MINimum><n>`**`?`**

Example: `Send(0,5,”:TRIGger:MINimum1?”,17,EOI);`
Response: <ASCII value>
Example: `+1.00123`

12-5 SEQUENCE

The **SEQUENCE** command selects the arming sequence between the START and STOP path. The two sequences are:

Arm on start
Arm on stop

The **SEQUENCE** query returns the presently selected arming sequence.

Command syntax- `:TRIGger:SEQuence<STARt|STOp>`

Example: `Send(0,5,”:TRIGger:SEQuenceSTARt”,22,EOI);`

Query syntax- `:TRIGger:SEQuence?`

Example: `Send(0,5,”:TRIGger:SEQuence?”,18,EOI);`
Response: `<Start|Stop>`

12-6 SLOPE

The **SLOPE** command sets the edge of a specific arming input. This edge can be a positive going (rising) edge or a negative going (falling) edge.

The **SLOPE** query returns the present setting of the specific external edge.

Command syntax- `:TRIGger:SLOPe<RISe|FALl>`

Example: `Send(0,5,”:TRIGger:SLOPeRIS”,17,EOI);`

Query syntax- `:TRIGger:SLOPe?`

Example: `Send(0,5,”:TRIGger:SLOPe?”,15,EOI) ;`
Response: `<RISe|FALl>`

12-7 SOURCE

The **SOURCE** command selects the arming signal that will initiate a measurement.

The **SOURCE** query returns the presently selected arming signal source.

The 2 source selections are:

`<EXTernal|AUTomatic>`

Command syntax- `:TRIGger:SOURce<EXTernal|AUTomatic>`

Example: `Send (0,5,”:TRIGger:SOURceEXTernal”,23,EOI);`

Query syntax- `:TRIGger:SOURce?`

Example: `Send (0,5,”:TRIGger:SOURce?”,16,EOI);`
Response: `<EXT|AUT>`

APPENDIX A - INTERNAL, EXTERNAL & STROBE CALIBRATION

This appendix describes all the programming steps to perform an Internal, Deskew and Strobe calibration from an host computer.

INTERNAL

The events required to perform an internal calibration:

Send the command to start internal calibration.
Respond with GO for message, “Ready to begin, select OK to continue…”.
Wait for TRG bit of a serial poll to indicate completion.
Check DDE bit of the ESR register to determine if an error occurred.

The code to implement the above events follows:

```
long perform_int_calibration (void)
  {
  char event_status;
  char poll_status;
  Send(0,5,”*CLS”,4,EOI);
  Send(0,5,”:CAL:INT”,8,EOI);
  respond_to_go_request(1);   /*continue int cal*/
  poll_status = 0;
  while ((poll_status&TRG) ==0)   {
    ReadStatusByte(0,5,&poll_status);
    }
  Send(0,5,”*ESR?”,5,EOI);
  Receive(0,5,event_status,1,EOI);
  if((event_status& DDE)!=Ø)
    {
    printf(“InternalCalibrationFailed”);
    return(-1);
    }
  else
    {
    printf(“InternalCalibrationPassed”);
    return(0);
    }
  }
```

DESKEW

The events required to perform an deskew calibration are:

Send command to start the deskew calibration
As prompted, respond with GO to swap cables between individual channels and between channel pairs or with NOGO to skip individual channels and/or channel pairs
Wait for TRG bit of a serial poll to indicate completion
Check DDE bit of the ESR register to determine if an error occurred

The code to implement the above events follows:

```
long perform_deskew (void)
  {
  /* Requires a timeout of 10 seconds (minimum) on the GPIB card */
  pMesg = (char **) 1; /* Deskew */

  char event_status;
  char poll_status;

  Send(0,5,”*CLS”,4,EOI);
  Send(0,5,”:CAL:DESKEW”,11,EOI);

  /* Loop through and deskew each individual channel */
  for ( chan = MIN_CHAN; chan <= MAX_CHAN; chan++ )
    {
    if ( respond_to_prompt ( 0, next ) )
      {
      printf(“Deskew Calibration Failed”);
      return(-1);
      }
    }

  /* Now do the channel to channel deskews */
  for ( chan = MIN_CHAN; chan <= (MAX_CHAN - 1); chan++ )
    {
    for ( pair = chan + 1; pair <= MAX_CHAN; pair++ )
      {
      if ( respond_to_prompt ( 0, next ) )
        {
        printf(“Deskew Calibration Failed”);
        return(-1);
        }
      }
    }
  poll_status = 0;
  while ((poll_status & TRG) == 0 )
```

```
    ReadStatusByte(0,5,&poll_status);
  Send(0,5,”*ESR?”,5,EOI);
  Receive(0,5,event_status,1,EOI);
  if ( (event_status & DDE) != 0) )
    {
    printf(“Deskew Calibration Failed”);
    return(-1);
    }
  else
    {
    printf(“Deskew Calibration Passed”);
    return(0);
    }
  }
```

DESKEW WITH DC OFFSET

The events required to perform an deskew calibration with DC offset are:

Send command to start the deskew calibration with DC offset
As prompted, respond with GO to perform the DC offset portion, swap cables between individual channels/channel pairs or with NOGO to skip individual channels and/or channel pairs
Wait for TRG bit of a serial poll to indicate completion
Check DDE bit of the ESR register to determine if an error occurred

The code to implement the above events follows:

```
long perform_deskew_with_dc (void)
  {
  /* Requires a timeout of 100 seconds (minimum) on the GPIB card */
  pMesg = (char **) 2; /* Deskew with DC calibration */

  char event_status;
  char poll_status;

  Send(0,5,”*CLS”,4,EOI);
  Send(0,5,”:CAL:DESKEWDC”,11,EOI);

  /* Loop through and deskew each individual channel */
  for ( chan = MIN_CHAN; chan <= MAX_CHAN; chan++ )
    {
    /* Perform DC portion of individual channel deskew */
    if ( respond_to_prompt ( 0, next ) )
        {
        printf(“Deskew (with DC) Calibration Failed”);
        return(-1);
      }
      if ( respond_to_prompt ( 0, next ) )
        {
        printf(“Deskew (with DC) Calibration Failed”);
        return(-1);
        }
      }

    /* Now do the channel to channel deskew */
    for ( chan = MIN_CHAN; chan <= (MAX_CHAN - 1); chan++ )
      {
      for ( pair = chan + 1; pair <= MAX_CHAN; pair++ )
        {
        if ( respond_to_prompt ( 0, next ) )
          {
          printf(“Deskew (with DC) Calibration Failed”);
          return(-1);
          }
        }
      }

    poll_status = 0;
    while ((poll_status & TRG) == 0 )
      ReadStatusByte(0,5,&poll_status);
    Send(0,5,”*ESR?”,5,EOI);
    Receive(0,5,event_status,1,EOI);
    if ( (event_status & DDE) != 0) )
      {
      printf(“Deskew (with DC) Calibration Failed”);
      return(-1);
      }
    else
      {
      printf(“Deskew (with DC) Calibration Passed”);
      return(0);
      }
    }


  #define MIN_CHAN  1  /* These values depend upon how many channels
  #define MAX_CHAN  5     have been installed in the SIA-3000     */

  static char **pMesg;
  static long nStep, nChan1, nChan2;

  long pPrompt ( void )
    {
```

```
int  rqst;
long retn;
char sPrompt[BUF_SIZ] = {0};

if (pMesg == (char **) 1)
   {
   /* This is the deskew calibration */
   if (nStep < MAX_CHAN)
      sprintf(sPrompt, “\n\nConnect the TOP Calibration Output to  the
                     TOP Ch%d Input,\nand the BOTTOM Calibration
                     output to the BOTTOM Ch%d Input.\nPress Y to
                     continue, N to skip. “, nStep + 1, nStep + 1);
   else
      {
      nChan2++;
      if (nChan2 == MAX_CHAN)
         {
         nChan1++;
         nChan2 = nChan1 + 1;
         }
         sprintf(sPrompt, “\n\nConnect the TOP Calibration Output to
                     the TOP Ch%d Input,\nand the BOTTOM Calibra
                     tion Output to the TOP Ch%d Input.\nPress Y
                     to continue, N to skip. “, nChan1 + 1, nChan2
                     + 1);
         }
      }
   else if (pMesg == (char **) 2)
      {
      /* This is the deskew calibration with DC */
      if (nStep < MAX_CHAN * 2)
         {
         if (nStep & 1)
            sprintf(sPrompt, “\n\nConnect the TOP Calibration Output to
                     the TOP Ch%d Input,\nand the BOTTOM Cali
                     bration Output to the BOTTOM Ch%d Input.
                     \nPress Y to continue, N to skip. “,
               nStep / 2 + 1, nStep / 2 + 1);
         else
            sprintf(sPrompt, “\n\nEnsure nothing is connected to Ch%d
                     inputs.\nPress Y to continue, N to skip. “,
               nStep / 2 + 1);
         }
      else
         {
         nChan2++;
      if (nChan2 == MAX_CHAN)
        {
        nChan1++;
        nChan2 = nChan1 + 1;
        }
      sprintf(sPrompt, “\n\nConnect the TOP Calibration Output to the TOP
                        Ch%d Input\nand the BOTTOM Calibration Output to the
                        TOP Ch%d Input.\nPress Y to continue, N to skip. “,
                        nChan1 + 1, nChan2 + 1);
      }
    }
  else
    sprintf(sPrompt, “\n\nError occurred during sequence. Press Q to
                      abort. “);

  printf(sPrompt);
  rqst = getch();
  rqst = toupper(rqst);
  printf(“%c”, rqst);

  /* Continue or skip? */
  if ( rqst == ‘Y’ )
    retn = 1;
  else
    retn = 0;

  /* This is for the deskew calibration with DC */
  if (pMesg == (char **) 2)
    {
    /* If we cancel on the first message skip the second */
    if (nStep < MAX_CHAN * 2)
      {
      if (retn == 0 && !(nStep & 1))
        nStep++;
      }
    }
  nStep++;
  return retn;
  }
```

```
long respond_to_prompt (long ( *next )( void ))
  {
  char poll_status;
  char event_status;

  event_status = 0;
  while ( (event_status & RQC) == 0 )
    {
    poll_status = 0;
    while ( (poll_status & ESB) == 0 )
      ReadStatusByte(0,5,&poll_status);
    Send(0,5,"*ESR?",5,EOI);
    Receive(0,5,event_status,1,EOI);
    }

  if ( next )
    {
    if ( !next() )
      Send(0,5,":SYST:NOGO",10,EOI);
    else
      Send(0,5,":SYST:GO",8,EOI);
    }
  return(0);
  }
```

STROBE

The events required to perform a strobe calibration are:

Ensure nothing is connected to any inputs.
Send the command to start strobe calibration.
Respond with GO for message, “Ensure nothing is connected to any inputs”.
Wait for TRG bit of a serial poll to indicate completion.
Check DDE bit of the ESR register to determine if an error occurred.

The code to implement the above events is as follows:

```
long perform_strobe_calibration (void)
  {
  char event_status;
  char poll_status;

  Send(0,5,"*CLS",4,EOI);
  Send(0,5,":SYST:STROCAL",13,EOI);

  /*start strobe cal*/
  respond_to_go_request(1);
```

```
        poll_status = 0;
        while ((poll_status&TRG) ==0)
          ReadStatusByte(0,5,&poll_status);
        Send(0,5,"*ESR?",5,EOI);
      Receive(0,5,event_status,1,EOI);
      if((event_status& DDE)!=Ø)
        {
        printf("Strobe Cal Failed");
        return(-1);
        }
      else
        {
        printf("Strobe Cal Passed");
        return(0);
        }
      }

    long respond_to_go_request (long go)
      {
      char poll_status;
      char event_status;

      event_status = 0;
      while ( (event_status & RQC) == 0 )
         {
         poll_status = 0;
         while ( (poll_status & ESB) == 0 )
           ReadStatusByte(0,5,&poll_status);
      Send(0,5,"*ESR?",5,EOI);
      Receive(0,5,event_status,1,EOI);
      }

      if ( !go )
         Send(0,5,":SYST:NOGO",10,EOI);
      else
         Send(0,5,":SYST:GO",8,EOI);
      return(0);
      }
```

APPENDIX B - READING DATA

This appendix describes the programming steps to take a measurement and read back the values of the measurement. In this example a burst of 100 measurements is taken and the data read back in a 32-bit floating format.

```
void main (void)
   {
      int i;
      int no_of_bytes;
      char temp_string[2048];
      int c;
      int header;
      float this_reading[100];
      char *ptr;
      int result;
/*488 is initialized for controller and instrument*/
   no_of_bytes=100;
   Send(0,5,"*CLS",4,EOI);
   Send(0,5,":ACQ:COUN100",12,EOI)
   Send(0,5,"*TRG",4,EOI);
   result=0;
   while((result & 0x01) ==0)      /*Wait for TRG bit*/
      {
      ReadStatusByte(0,5,&result);
      }
   sprintf(temp_string,"%i",number_of_bytes*4);
   c=strlen(temp_string);
   header=c+2;                     /*# of characters in header*/
   Send(0,5,":MEAS:DATA4?",12,EOI);
   result=0;
   while((result & 0x10) ==0)      /*Wait for MAV bit*/
      {
      ReadStatusByte(0,5,&result);
      }
   Receive(0,5,temp_string,(no_of_bytes*4)+header,EOI);
                                   /*convert char string to floating point*/
   ptr=&temp_string [header];
   for(i=0; i<no_of bytes; i++)
      {
      this_reading [i] = *((float*)ptr);
      ptr = ptr+4;
      }
   }                               /*end of main*/
```

This page intentionally left blank.

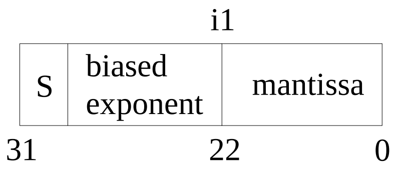
APPENDIX C - DATA TYPES

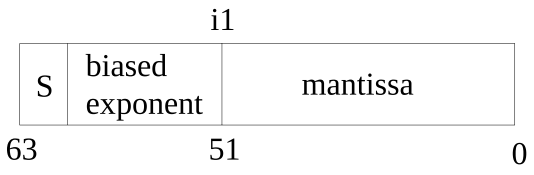
This appendix describes the data formats used for transferring data from the SIA-3000 over the
GPIB for **:MEAS**ure commands.

The **:MEAS**ure**:DATA**, **:MEAS**ure**:DATA4** and **:MEAS**ure**:DATAT** queries support two sizes of data types using IEEE standards for floating-point arithmetic (ANSI/IEEE Std. 754-1985):

Type	Size (bits)	Smallest Absolute value	Largest Absolute value	Number of Digit Accuracy
float	32	$1.1\text{x}10^{-38}$	$3.4\text{x}10^{38}$ Scientific	6-digit precision
double	64	$2.2\text{x}10^{-308}$	$1.7\text{x}10^{308}$ Scientific	15-digit precision

Data Representation:





s = Signbit (0 = positive, 1 = negative)
i = Position of implicit binary point (always 1)
1 = integer bit of mantissa
Exponent bias (normalized values)
float: 127(7FH)
double: 1023(3FFH)

This page intentionally left blank.

WAVECREST CORPORATION

World Headquarters:
7626 Golden Triangle Drive
Eden Prairie, MN 55344
TEL: (952) 831-0030
FAX: (952) 831-4474
Toll Free: 1-800-733-7128
www.wavecrest.com

West Coast Office:
1735 Technology Drive, Ste. 400
San Jose, CA 95110
TEL: (408) 436-9000
FAX: (408) 436-9001
1-800-821-2272

Europe Office:
Lilenthalalle 25
D-80939 Munchen
TEL: 011-49-89-32225330
FAX: 011-49-89-32225333

Japan Office:
Otsuka Sentcore Building, 6F
3-46-3 Minami-Otsuka
Toshima-Ku, Tokyo
170-0005, Japan
TEL: +81-03-5960-5770
Fax: +81-03-5960-5773

200007-00 REV A