Agilent 6000 Series Oscilloscopes Programmer's Reference

Version 04.00.0000

Warranty
Technology License
Restricted Rights Legend
Safety Notices
Trademarks

January 5, 2007

Table of Contents

Agilent Technologies

In This Book

This programmer's reference gives detailed information on all the commands available for controlling these oscilloscope models:

6000 Series Oscilloscope Models:

	Input Bandwidth			
Channels	1 GHz	500 MHz	300 MHz	100 MHz
4 analog + 16 digital (mixed-signal)	MSO6104A/L	MSO6054A/L	MSO6034A	MSO6014A/L
2 analog + 16 digital (mixed-signal)	MSO6102A	MSO6052A	MSO6032A	MSO6012A
4 analog	DSO6104A/L	DSO6054A/L	DSO6034A	DSO6014A/L
2 analog	DSO6102A	DSO6052A	DSO6032A	DSO6012A

The command descriptions in this reference show upper and lowercase characters. For example, :AUToscale indicates that the entire command name is :AUTOSCALE. The short form, :AUT, is also accepted by the oscilloscope.

Command arguments and syntax are described for each command. Some command descriptions have example code.

See the *Agilent 6000 Series Oscilloscopes Programmer's Quick Start Guide* for information on installing the IO libraries, connecting the oscilloscope to the controller PC, and getting started with oscilloscope programming.

See your oscilloscope's *User's Guide* for more information on front-panel operation.

Mixed-Signal Oscilloscope Channel Differences

Because both the "analog channels only" oscilloscopes (DSO models) and the mixed-signal oscilloscopes (MSO models) have analog channels, topics that describe analog channels refer to all oscilloscope models. Whenever a topic describes digital channels, that information applies only to the mixed-signal oscilloscope models.

Example Programs

The example programs are designed to work with multiple 6000 Series oscilloscopes. Therefore, the commands may not match the example code exactly, but the example code should run because of the designed-in backward compatibility with earlier commands.

Agilent 6000 Series Oscilloscopes Programmer's Reference

What's New

What's New

What's New in Version 4.00

New features in version 4.00 of the 6000 Series oscilloscope software are:

- The ability to :AUToscale selected channels only and specify the acquisition type and mode that is set after an :AUToscale.
- The :BUS command subsystem for controlling up to two buses made up of digital channels.
- Additional :CALibrate commands for starting the user calibration procedure, displaying the status of the last user calibration, and displaying the temperature change since the last user calibration.

More detailed descriptions of the new and changed commands appear below.

New Commands

Command	Description
:AUToscale:AMODE	Specifies whether to keep the current acquisition type and mode after subsequent autoscales.
:AUToscale:CHANnels	Specifies whether to autoscale the currently displayed channels or all channels.
:BUS<n>:BIT<m>	Includes or excludes the selected bit in a bus definition.
:BUS<n>:BITS	Includes or excludes a list of bits in a bus definition.
:BUS<n>:CLEar	Excludes all digital channels from a bus definition
:BUS<n>:DISPlay	Displays or hides the bus on the oscilloscope display.

:BUS<n>:LABel	Assigns a label string to a bus.
:BUS<n>:MASK	Includes or excludes bits in a bus definition according to a mask.
:CALibrate:STARt	Starts the user calibration procedure.
:CALibrate:STATus?	Displays the summary results of the last user calibration procedure.
:CALibrate:TEMPerature?	Displays the change in temperature since the last user calibration procedure.

Changed Commands

Command	Differences
:AUToscale	You can now specify which channels to autoscale.
:BLANk	Now, you can also use this command with digital channel buses.
:DIGitize	Now, you can also use this command with digital channel buses.
:STATus	Now, you can also use this command with digital channel buses.
:VIEW	Now, you can also use this command with digital channel buses.
:WAVeform:SOURce	Now, you can also use this command with digital channel buses.

What's New

What's New in Version 3.50

New features in version 3.50 of the 6000 Series oscilloscope software are:

- The CAN and LIN options have been added to the :SBUS:MODE (serial decode mode) command.
- The :SBUS:CAN:COUNt commands have been added to count CAN bus frames, count load utilization, and reset the counters.
- The ALLerrors, OVERload, and ACKerror options have been added to the :TRIGger:CAN:TRIGger command.
- The :TRIGger:LIN:ID, :TRIGger:LIN:SAMPlepoint, :TRIGger:LIN:STANdard, and :TRIGger:LIN:SYNCbreak commands have been added.
- The :SBUS:LIN:PARity command has been added.
- The ID (for Frame Id) option has been added to the :TRIGger:LIN:TRIGger command.
- The :HWERegister:CONDition, :HWERegister[:EVENt], and :HWE commands for the hardware event condition, event, and enable registers have been added.

More detailed descriptions of the new and changed commands appear below.

New Commands

Command	Description
:SBUS:CAN:COUNt:ERRor?	Returns the CAN bus error frame count.
:SBUS:CAN:COUNt:OVERload?	Returns the CAN bus overload frame count.
:SBUS:CAN:COUNt:RESet	Resets the CAN bus counters.

:SBUS:CAN:COUNt:TOTal?	Returns the CAN bus total frame count.
:SBUS:CAN:COUNt:UTILization?	Returns a percentage showing CAN bus utilization.
:SBUS:IIC:ASIZe	Determines whether the Read/Write bit is included as the LSB in the display of the IIC address field of the decode bus.
:SBUS:LIN:PARity	Determines whether the parity bits are included as the most significant bits (MSB) in the display of the Frame Id field in the LIN decode bus.
:TRIGger:LIN:ID	Defines the LIN identifier searched for in each CAN message when the LIN trigger mode is set to frame ID.
:TRIGger:LIN:SAMPlepoint	Sets the point during the bit time where the bit level is sampled to determine whether the bit is dominant or recessive. The sample point represents the percentage of time between the beginning of the bit time to the end of the bit time.
:TRIGger:LIN:STANdard	Sets the LIN standard in effect for triggering and decoding to be LIN1.3 or LIN2.0.
:TRIGger:LIN:SYNCbreak	Sets the length of the LIN sync break to be greater than or equal to 11,12, or 13 clock lengths. The sync break is the idle period in the bus activity at the beginning of each packet that distinguishes one information packet from the previous one.
:HWEenable	Sets or reads the hardware event enable mask register.
:HWERegister:CONDition?	Queries the hardware event condition register.
:HWERegister[:EVENt]?	Queries the hardware event event register.

Changed Commands

Command	Differences
:SBUS:MODE	The CAN and LIN serial bus decode modes have been added.
:TRIGger:CAN:TRIGger	The ALLerrors, OVERload, and ACKerror options have been added.
:TRIGger:LIN:TRIGger	The ID (for Frame Id) option has been added.

Obsolete Commands

Obsolete Command	Current Command Equivalent	Behavior Differences
:TRIGger:CAN:SIGNal:DEFinition	none	
:TRIGger:LIN:SIGNal:DEFinition	none	

What's New

What's New in Version 3.00

New features in version 3.00 of the 6000 Series oscilloscope software are:

- The :SBUS command subsystem for controlling serial decode bus display, mode, and other options.
- The EBURst trigger mode and supporting :TRIGger:EBURst commands.

- The :ACQuire:AALias and :ACQuire:DAALias commands.
- The :WAVeform:POINts:MODE command.
- The :MEASure:SDEViation command.
- The :TIMebase:REFClock command.
- Changes to the :TRIGger:IIC commands.
- Changes to the :TRIGger:SEQuence:TRIGger command.
- Changes to the :ACQuire:TYPE and :WAVeform:TYPE commands to add HRESolution type.
- Changes to the :BLANk, :DIGitize, :STATus, :VIEW, and :WAVeform:SOURce commands to include the serial decode bus.
- Changes to the :HARDcopy:FORMat command to support the PNG, ASCiixy, and BINary format types.
- Changes to the :DISPlay:DATA? query and the :PRINt command to support the PNG format.
- Changes to the :WAVeform:POINts command to set from 2000 to 8,000,000 points (in 1-2-5 sequence) when the waveform points mode is MAXimum or RAW.

More detailed descriptions of the new and changed commands appear below.

New Commands

Command	Description
:ACQuire:AALias?	Returns the current state of the oscilloscope's anti-alias control.
:ACQuire:DAALias	Sets the oscilloscope's disable anti-alias mode.
:MEASure:SDEViation	Measures the std deviation of a waveform.
:SBUS:DISPlay	Controls the decoded serial bus display.
:SBUS:MODE	Determines the decode mode for the serial bus.
:SBUS:SPI:WIDTh	Determines the number of bits in a word of decoded data for SPI.
:TIMebase:REFClock	Enables or disables the 10 MHz REF BNC input/output.
:TRIGger:EBURst:COUNt	Sets the Nth edge of burst edge counter resource.
:TRIGger:EBURst:IDLE	Sets the Nth edge in a burst idle resource.
:TRIGger:EBURst:SLOPe	Specifies whether the rising edge (POSitive) or falling edge (NEGative) of the Nth edge in a burst will generate a trigger.
:TRIGger:IIC:PATTern:DATa2	Sets IIC data 2.
:WAVeform:POINts:MODE	Sets the waveform points mode.

Changed Commands

Command	Differences
:ACQuire:TYPE	The HRESolution type has been added for smoothing at slower sweep speeds.
:BLANk	Now, you can also use this command with the serial decode bus.
:DIGitize	Now, you can also use this command with the serial decode bus.
:DISPlay:DATA	Now, the PNG format is supported in the query.

:HARDcopy:FORMat	Now, the PNG, ASCiixy, and BINary formats are also supported.
:PRINt	Now, the PNG option is supported
:STATus	Now, you can also use this command with the serial decode bus.
:TRIGger:IIC:TRIGger [:TYPE]	The ANACknowledge, R7Data2, and W7Data2 types have been added.
:TRIGger:MODE	The EBURst mode has been added.
:TRIGger:SEQuence:TRIGger	The EDGE2,COUNt,NREFind (no re-find) option has been added.
:VIEW	Now, you can now use this command with the serial decode bus.
:WAVeform:POINts	Now, you can set from 2000 to 8,000,000 points (in 1-2-5 sequence) when the waveform points mode is MAXimum or RAW.
:WAVeform:SOURce	Now, you can also use this command with the serial decode bus.
:WAVeform:TYPE	The HRESolution type has been added for smoothing at slower sweep speeds.

What's New

Command Differences From 54620/54640 Series Oscilloscopes

The main differences between the version 1.00 programming command set for the 6000 Series oscilloscopes and the 54620/54640 Series oscilloscopes are related to:

- :HARDcopy and :DISPlay command subsystem changes for USB printers and the high resolution color display.
- New standards supported by the :TRIGger:TV commands.
- Support for 113xA Series probes.
- New "RAW" :WAVeform:POINts option for retrieving raw acquisition record data.
- Discontinuance of the common commands for macros.

More detailed descriptions of the new, changed, obsolete, and discontinued commands appear below.

New Commands

Command	Description
:ACQuire:RSIGnal	Selects the 10 MHz reference signal mode.
:CHANnel<n>:PROBe:ID?	Returns the type of probe attached to the specified oscilloscope channel.
:CHANnel<n>:PROBe:STYPe	Sets the channel probe signal type (STYPe) to differential or single-ended when using the 113xA Series probes, and determines how offset is applied.
:CHANnel<n>:VERNier	Specifies whether the channel's vernier (fine vertical adjustment) setting is ON (1) or OFF (0).
:DIGital<n>:SIZE	Specifies the size of digital channels on the display.
:EXTernal:PROBe:ID	Returns the type of probe attached to the external trigger input.
:EXTernal:PROBe:STYPe	Sets the external trigger probe signal type (STYPe) to differential or single-ended when using the 113xA Series probes, and determines how offset is

	applied.
:HARDcopy:FILename	Sets the output filename for print formats whose output is a file. Replaces the 5462x/4x :HARDcopy:DESTination command.
:HARDcopy:PDRiver	Sets the hardcopy printer driver.
:HARDcopy:IGColors	Specifies whether graticule colors are inverted.
:HARDcopy:PALette	Sets the hardcopy palette color. Replaces the 5462x/4x :HARDcopy:GRAYscale command.
:OPERegister:CONDition?	Returns the integer value contained in the Operation Status Condition Register (a new register in addition to the Operation Status Event Register whose value is returned by the :OPERegister[:EVENt]? query).
:POD<n>:SIZE	Specifies the size of digital channels on the display.
:TIMebase:VERNier	Specifies whether the time base control's vernier (fine horizontal adjustment) setting is ON (1) or OFF (0).

Changed Commands

Command	Differences From 5462x/4x Oscilloscopes
:ACQuire:COUNt	The count can be set to any value from 1 to 65536 (instead of 16383).
:DISPlay:DATA	The BMP8bit <format> option has been added to the query. There is a new <palette> option which can be MONochrome, GRAYscale, or COLor in the query, or just MONochrome in the command.
:DISPlay:SOURce	The number of pixel memory locations is 10 (instead of 3).
:HARDcopy:FORMat	There is now the BMP8bit format (instead of TIFF) and the PRINter0 or PRINter1 formats (in place of LASerjet, DESKjet, EPSon, or SEIKo). See the new :HARDcopy:PDRiver command for setting the hardcopy printer driver.
*LRN	The Learn Device Setup query return format matches the IEEE 488.2 specification which says that the query result must contain ":SYST:SET " before the binary block data. (This was not the case in the 5462x/4x oscilloscopes.)
:MERGe	The number of pixel memory locations is 10 (instead of 3).
*OPT	The Option Identification query return format now has license information (in addition to the I/O module ID information fields which are now always zero).
:OVLRegister	The Overload Event Register is now a 16-bit register (instead of 8-bit) and it contains bits that identify when faults occur on the oscilloscope channels (in addition to the bits that identify when overloads occur).
:PRINt	The options are now: COLor (instead of HIRes), GRAYscale (instead of LORes), PRINter0 (instead of PARallel), BMP8bit (instead of TIFF). (The PCL option is now invalid.)
*RCL (Recall)	The number of instrument state locations is 10 (instead of 3 for the 54620 Series oscilloscopes or 4 for the 54640 Series oscilloscopes).
*SAV (Save)	The number of instrument state locations is 10 (instead of 3 for the 54620 Series oscilloscopes or 4 for the 54640 Series oscilloscopes).
*TRG (Trigger)	The *TRG has the same effect as the :DIGitize command with no parameters (instead of the :RUN command).
:TRIGger:TV:MODE	The modes have been renamed (however, old forms of the mode names are still

	accepted).
:TRIGger:TV:STANdard	The P480L60HZ, P720L60HZ, P1080L24HZ, P1080L25HZ, I1080L50HZ, and I1080L60HZ standards are supported (in addition to GENeric, NTSC, PALM, PAL, and SECam).
:VIEW	The number of pixel memory locations is 10 (instead of 3).
:WAVeform:COUNt?	The count can be any value from 1 to 65536 (instead of 16383).
:WAVeform:POINts	There is a new RAW "number of points" option for retrieving the raw acquisition record data. Also the maximum number of points that can be retrieved from the normal measurement record is 1000 (instead of 2000).
:WAVeform:PREamble	The xincrement format is 64-bit floating point NR3 (instead of 32-bit), and the yreference format is 32-bit NR1 (instead of 16-bit).
:WAVeform:XINCrement	The x-increment value from the preamble is returned in 64-bit (instead of 32-bit) floating point NR3 format.
:WAVeform:YREFerence	The y-reference value from the preamble is returned in 32-bit (instead of 16-bit) NR1 format.

Obsolete Commands

Obsolete Command	Current Command Equivalent	Behavior Differences
:HARDcopy:DESTination	:HARDcopy:FILename	
:HARDcopy:GRAYscale	:HARDcopy:PALette	
:PRINt?	:DISPlay:DATA?	The options are now: COLor (instead of HIRes), GRAYscale (instead of LORes), PRINter0 (instead of PARallel), BMP8bit (instead of TIFF). (The DISK and PCL options are now invalid.)

Discontinued Commands

Command	Description
*DMC	Define Macro.
*EMC	Enable Macro.
*GMC	Get Macro Contents.
*LMC	Learn Macro.
*PMC	Purge Macro.

Agilent 6000 Series Oscilloscopes Programmer's Reference

Commands Quick Reference

Command	Query	Options and Query Returns
n/a	:ACQuire:AALias?	{1 | 0}
:ACQuire:COMPlete <complete>	:ACQuire:COMPlete?	<complete> ::= 100; an in NR1 format

:ACQuire:COUNt <count>	:ACQuire:COUNt?	<count> ::= an integer fr 65536 in NR1 format
:ACQuire:DAALias <mode>	:ACQuire:DAALias?	<mode> ::= {DISable | AUT
:ACQuire:MODE <mode>	:ACQuire:MODE?	<mode> ::= {RTIMe | ETIMe
n/a	:ACQuire:POINts?	<# points> ::= an integer format
:ACQuire:RSIGnal <ref_signal_mode>	:ACQuire:RSIGnal?	<ref_signal_mode> ::= {OF | IN}
n/a	:ACQuire:SRATe?	<sample_rate> ::= sample (samples/s) in NR3 format
:ACQuire:TYPE <type>	:ACQuire:TYPE?	<type> ::= {NORMal | AVER HRESolution | PEAK}
:ACTivity	:ACTivity?	<return value> ::= <edges>,<levels> <edges> ::= presence of e (32-bit integer in NR1 fo <levels> ::= logical high lows (32-bit integer in N format)
n/a	:AER?	{0 | 1}; an integer in NR
:AUToscale [<source> [,..,<source>]]	n/a	<source> ::= CHANnel<n> f models <source> ::= {CHANnel<n> DIGital0,..,DIGital15 | P POD2} for MSO models <source> can be repeated times <n> ::= 1-2 or 1-4 in NR1
:AUToscale:AMODE <value>	:AUToscale:AMODE?	<value> ::= {NORMal | CUR
:AUToscale:CHANnels <value>	:AUToscale:CHANnels?	<value> ::= {ALL | DISPla
:BLANk [<source>]	n/a	<source> ::= {CHANnel<n>} FUNCtion | MATH | SBUS} f models <source> ::= {CHANnel<n> DIGital0,..,DIGital15 | P 2} | BUS{1 | 2} | FUNCtio | SBUS} for MSO models <n> ::= 1-2 or 1-4 in NR1
:BUS<n>:BIT<m> {{0 | OFF} | {1 | ON}}	:BUS<n>:BIT<m>?	{0 | 1} <n> ::= 1 or 2; an intege format <m> ::= 0-15; an integer

		format
:BUS<n>:BITS <channel_list>, {{0 | OFF} | {1 | ON}}	:BUS<n>:BITS?	<channel_list>, {0 | 1} <channel_list> ::= (@<m>,<m>:<m> ...) where separator and ":" is rang <n> ::= 1 or 2; an intege format <m> ::= 0-15; an integer format
:BUS<n>:CLEar	n/a	<n> ::= 1 or 2; an intege format
:BUS<n>:DISPlay {{0 | OFF} | {1 | ON}}	:BUS<n>:DISPlay?	{0 | 1} <n> ::= 1 or 2; an intege format
:BUS<n>:LABel <string>	:BUS<n>:LABel?	<string> ::= quoted ASCII up to 16 characters <n> ::= 1 or 2; an intege format
:BUS<n>:MASK <mask>	:BUS<n>:MASK?	<mask> ::= 32-bit integer decimal, <nondecimal>, or <string> <nondecimal> ::= #Hnn...n n ::= {0,..,9 | A,..,F} f hexadecimal <nondecimal> ::= #Bnn...n n ::= {0 | 1} for binary <string> ::= "0xnn...n" w n ::= {0,..,9 | A,..,F} f hexadecimal <n> ::= 1 or 2; an intege format
n/a	:CALibrate:DATE?	<return value> ::= <day>,<month>,<year>; all format
:CALibrate:LABel <string>	:CALibrate:LABel?	<string> ::= quoted ASCII up to 32 characters
:CALibrate:STARt	n/a	n/a
n/a	:CALibrate:STATus?	<return value> ::= ALL,<status_code>,<status <status_code> ::= an inte status code <status_string> ::= an AS

		status string
n/a	:CALibrate:SWITch?	{PROTected | UNPROtected}
n/a	:CALibrate:TEMPerature?	<return value> ::= degree delta since last cal in N format
n/a	:CALibrate:TIME?	<return value> ::= <hours>,<minutes>,<second in NR1 format
:CDISplay	n/a	n/a
:CHANnel<n>:BWLimit {{0 | OFF} | {1 | ON}}	:CHANnel<n>:BWLimit?	{0 | 1} <n> ::= 1-2 or 1-4 in NR1
:CHANnel<n>:COUPling <coupling>	:CHANnel<n>:COUPling?	<coupling> ::= {AC | DC} <n> ::= 1-2 or 1-4 in NR1
:CHANnel<n>:DISPlay {{0 | OFF} | {1 | ON}}	:CHANnel<n>:DISPlay?	{0 | 1} <n> ::= 1-2 or 1-4 in NR1
:CHANnel<n>:IMPedance <impedance>	:CHANnel<n>:IMPedance?	<impedance> ::= {ONEMeg | <n> ::= 1-2 or 1-4 in NR1
:CHANnel<n>:INVert {{0 | OFF} | {1 | ON}}	:CHANnel<n>:INVert?	{0 | 1} <n> ::= 1-2 or 1-4 in NR1
:CHANnel<n>:LABel <string>	:CHANnel<n>:LABel?	<string> ::= any series o less ASCII characters enc quotation marks <n> ::= 1-2 or 1-4 in NR1
:CHANnel<n>:OFFSet <offset> [suffix]	:CHANnel<n>:OFFSet?	<offset> ::= Vertical off value in NR3 format [suffix] ::= {V | mV} <n> ::= 1-2 or 1-4; in NR
:CHANnel<n>:PROBe <attenuation>	:CHANnel<n>:PROBe?	<attenuation> ::= Probe attenuation ratio in NR3 <n> ::= 1-2 or 1-4 in NR1
n/a	:CHANnel<n>:PROBe:ID?	<probe id> ::= unquoted A string up to 11 character <n> ::= 1-2 or 1-4 in NR1
:CHANnel<n>:PROBe:SKEW <skew_value>	:CHANnel<n>:PROBe:SKEW?	<skew_value> ::= -100 ns ns in NR3 format <n> ::= 1-2 or 1-4 in NR1
:CHANnel<n>:PROBe:STYPe <signal	:CHANnel<n>:PROBe:STYPe?	<signal type> ::= {DIFFer

type>		SINGle} <n> ::= 1-2 or 1-4 in NR1
:CHANnel<n>:PROTection	:CHANnel<n>:PROTection?	{NORM | TRIP} <n> ::= 1-2 or 1-4 in NR1
:CHANnel<n>:RANGe <range> [suffix]	:CHANnel<n>:RANGe?	<range> ::= Vertical full range value in NR3 format [suffix] ::= {V | mV} <n> ::= 1-2 or 1-4 in NR1
:CHANnel<n>:SCALe <scale> [suffix]	:CHANnel<n>:SCALe?	<scale> ::= Vertical unit division value in NR3 for [suffix] ::= {V | mV} <n> ::= 1-2 or 1-4 in NR1
:CHANnel<n>:UNITs <units>	:CHANnel<n>:UNITs?	<units> ::= {VOLTs | AMPe <n> ::= 1-2 or 1-4 in NR1
:CHANnel<n>:VERNier {{0 | OFF} | {1 | ON}}	:CHANnel<n>:VERNier?	{0 | 1} <n> ::= 1-2 or 1-4 in NR1
*CLS	n/a	n/a
:DIGital<n>:DISPlay {{0 | OFF} | {1 | ON}}	:DIGital<n>:DISPlay?	{0 | 1} <n> ::= 0-15; an integer format
:DIGital<n>:LABel <string>	:DIGital<n>:LABel?	<string> ::= any series o less ASCII characters enc quotation marks <n> ::= 0-15; an integer format
:DIGital<n>:POSition <position>	:DIGital<n>:POSition?	<n> ::= 0-15; an integer format <position> ::= 0-7 if dis size = large, 0-15 if siz medium, 0-31 if size = sm
:DIGital<n>:SIZE <value>	:DIGital<n>:SIZE?	<value> ::= {SMALl | MEDi LARGe}
:DIGital<n>:THReshold <value> [suffix]	:DIGital<n>:THReshold?	<n> ::= 0-15; an integer format <value> ::= {CMOS | ECL | <user defined value>} <user defined value> ::= NR3 format from -8.00 to

		[suffix] ::= {V | mV | uV
:DIGitize [<source>[,..,<source>]]	n/a	<source> ::= {CHANnel<n> | FUNCtion | MATH | SBUS} f models <source> ::= {CHANnel<n> | DIGital0,..,DIGital15 | P 2} | BUS{1 | 2} | FUNCtio | SBUS} for MSO models <source> can be repeated times <n> ::= 1-2 or 1-4 in NR1
:DISPlay:CLEar	n/a	n/a
:DISPlay:DATA [<format>][,][<area>][,][<palette>]<display data>	:DISPlay:DATA? [<format>][,][<area>][,][<palette>]	<format> ::= {TIFF} (comm <area> ::= {GRATicule} (c <palette> ::= {MONochrome (command) <format> ::= {TIFF | BMP BMP8bit | PNG} (query) <area> ::= {GRATicule | S (query) <palette> ::= {MONochrome GRAYscale | COLor} (query <display data> ::= data i 488.2 # format
:DISPlay:LABel {{0 | OFF} | {1 | ON}}	:DISPlay:LABel?	{0 | 1}
:DISPlay:LABList <binary block>	:DISPlay:LABList?	<binary block> ::= an ord list of up to 75 labels, characters maximum, separ newline characters
:DISPlay:PERSistence <value>	:DISPlay:PERSistence?	<value> ::= {MINimum | IN
:DISPlay:SOURce <value>	:DISPlay:SOURce?	<value> ::= {PMEMory{0 | 3 | 4 | 5 | 6 | 7 | 8 | 9
:DISPlay:VECTors {{1 | ON} | {0 | OFF}}	:DISPlay:VECTors?	{1 | 0}
*ESE <mask>	*ESE?	<mask> ::= 0 to 255; an i in NR1 format: Bit Weight Name Enables 7 128 PON Power O 6 64 URQ User Re

5	32	CME	Command
4	16	EXE	Executi Error
3	8	DDE	Device Depende Error
2	4	QYE	Query E
1	2	RQL	Request Control
0	1	OPC	Operati Complet

n/a	*ESR?	<status> ::= 0 to 255; an in NR1 format
:EXTernal:BWLimit <bwlimit>	:EXTernal:BWLimit?	<bwlimit> ::= {0 | OFF}
:EXTernal:IMPedance <value>	:EXTernal:IMPedance?	<impedance> ::= {ONEMeg |
:EXTernal:PROBe <attenuation>	:EXTernal:PROBe?	<attenuation> ::= probe attenuation ratio in NR3
n/a	:EXTernal:PROBe:ID?	<probe id> ::= unquoted A string up to 11 character
:EXTernal:PROBe:STYPe <signal type>	:EXTernal:PROBe:STYPe?	<signal type> ::= {DIFFer SINGle}
:EXTernal:PROTection[:CLEar]	:EXTernal:PROTection?	{NORM | TRIP}
:EXTernal:RANGe <range> [<suffix>]	:EXTernal:RANGe?	<range> ::= vertical full range value in NR3 format <suffix> ::= {V | mV}
:EXTernal:UNITs <units>	:EXTernal:UNITs?	<units> ::= {VOLTs | AMPe
:FUNCtion:CENTer <frequency>	:FUNCtion:CENTer?	<frequency> ::= the curre center frequency in NR3 f The range of legal values 0 Hz to 25 GHz.
:FUNCtion:DISPlay {{0 | OFF} | {1 | ON}}	:FUNCtion:DISPlay?	{0 | 1}
:FUNCtion:OFFSet <offset>	:FUNCtion:OFFSet?	<offset> ::= the value at screen in NR3 format. The range of legal values 10 times the current sens of the selected function.
:FUNCtion:OPERation <operation>	:FUNCtion:OPERation?	<operation> ::= {SUBTract MULTiply | INTegrate | DIFFerentiate | FFT}
:FUNCtion:RANGe <range>	:FUNCtion:RANGe?	<range> ::= the full-scal vertical axis value in NR format. The range for ADD, SUBT,

		8E-6 to 800E+3. The range INTegrate function is 8E- 400E+3. The range for the DIFFere function is 80E-3 to 8.0E (depends on current sweep The range for the FFT fun 8 to 800 dBV.
:FUNCtion:REFerence <level>	:FUNCtion:REFerence?	<level> ::= the current r level in NR3 format. The range of legal values 400.0 dBV to +400.0 dBV (depending on current ran value).
:FUNCtion:SCALe <scale value> [<suffix>]	:FUNCtion:SCALe?	<scale value> ::= integer format <suffix> ::= {V | dB}
:FUNCtion:SOURce <source>	:FUNCtion:SOURce?	<source> ::= {CHANnel<n> SUBT | MULT} <n> ::= 1-2 or 1-4 in NR1
:FUNCtion:SPAN <span>	:FUNCtion:SPAN?	<span> ::= the current fr span in NR3 format. Legal values are 1 Hz to
:FUNCtion:WINDow <window>	:FUNCtion:WINDow?	<window> ::= {RECTangular HANNing | FLATtop}
:HARDcopy:FACTors {{0 | OFF} | {1 | ON}}	:HARDcopy:FACTors?	{0 | 1}
:HARDcopy:FFEed {{0 | OFF} | {1 | ON}}	:HARDcopy:FFEed?	{0 | 1}
:HARDcopy:FILename <string>	:HARDcopy:FILename?	<string> ::= quoted ASCII
:HARDcopy:FORMat <format>	:HARDcopy:FORMat?	<format> ::= {BMP[24bit] BMP8bit | PNG | CSV | ASC BINary | PRINter0 | PRINt
:HARDcopy:IGColors {{0 | OFF} | {1 | ON}}	:HARDcopy:IGColors?	{0 | 1}
:HARDcopy:PALette <palette>	:HARDcopy:PALette?	<palette> ::= {COLor | GR
:HARDcopy:PDRiver <driver>	:HARDcopy:PDRiver?	<driver> ::= {AP2Xxx | AP {AP2560 | AP25} | {DJ350 | DJ6xx | {DJ630 | DJ63} DJ6Special | DJ6Photo | DJ8Special | DJ8xx | DJ9V DJ9xx | GVIP | {PS100 | P CLASer | MLASer | POSTscr
:HWEenable <n>	:HWEenable?	<n> ::= 16-bit integer in format

n/a	:HWERregister:CONDition?	<n> ::= 16-bit integer in format
n/a	:HWERegister[:EVENt]?	<n> ::= 16-bit integer in format
n/a	*IDN?	AGILENT TECHNOLOGIES,<model>,<ser number>,X.XX.XX <model> ::= the model num the instrument <serial number> ::= the s number of the instrument <X.XX.XX> ::= the softwar revision of the instrumen
n/a	*LRN?	<learn_string> ::= curren instrument setup as a blo data in IEEE 488.2 # form
:MARKer:MODE <mode>	:MARKer:MODE?	<mode> ::= {OFF | MEASure MANual}
:MARKer:X1Position <position> [suffix]	:MARKer:X1Position?	<position> ::= X1 cursor value in NR3 format [suffix] ::= {s | ms | us ps | Hz | kHz | MHz} <return_value> ::= X1 cur position value in NR3 for
:MARKer:X1Y1source <source>	:MARKer:X1Y1source?	<source> ::= {CHANnel<n> FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 <return_value> ::= <sourc
:MARKer:X2Position <position> [suffix]	:MARKer:X2Position?	<position> ::= X2 cursor value in NR3 format [suffix] ::= {s | ms | us ps | Hz | kHz | MHz} <return_value> ::= X2 cur position value in NR3 for
:MARKer:X2Y2source <source>	:MARKer:X2Y2source?	<source> ::= {CHANnel<n> FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 <return_value> ::= <sourc
n/a	:MARKer:XDELta?	<return_value> ::= X curs delta value in NR3 format
:MARKer:Y1Position <position> [suffix]	:MARKer:Y1Position?	<position> ::= Y1 cursor value in NR3 format

		[suffix] ::= {V | mV | dB <return_value> ::= Y1 cur position value in NR3 for
:MARKer:Y2Position <position> [suffix]	:MARKer:Y2Position?	<position> ::= Y2 cursor value in NR3 format [suffix] ::= {V | mV | dB <return_value> ::= Y2 cur position value in NR3 for
n/a	:MARKer:YDELta?	<return_value> ::= Y curs delta value in NR3 format
:MEASure:CLEar	n/a	n/a
:MEASure:COUNter [<source>]	:MEASure:COUNter? [<source>]	<source> ::= {CHANnel<n>} models <source> ::= {CHANnel<n> DIGital0,..,DIGital15} fo models <n> ::= 1-2 or 1-4 in NR1 <return_value> ::= counte frequency in Hertz in NR3
:MEASure:DEFine DELay, <delay spec>	:MEASure:DEFine? DELay	<delay spec> ::= <edge_spec1>,<edge_spec2> edge_spec1 ::= [<slope>] <occurrence> edge_spec2 ::= [<slope>] <occurrence> <slope> ::= {+ | -} <occurrence> ::= integer

:MEASure:DEFine THResholds, <threshold spec>	:MEASure:DEFine? THResholds	<threshold spec> ::= {STA {<threshold mode>,<upper> <middle>,<lower>}
		<threshold mode> ::= {PER ABSolute}

:MEASure:DELay [<source1>] [,<source2>]	:MEASure:DELay? [<source1>] [,<source2>]	<source1,2> ::= {CHANnel< FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 <return_value> ::= floati number delay time in seco NR3 format
:MEASure:DUTYcycle [<source>]	:MEASure:DUTYcycle? [<source>]	<source> ::= {CHANnel<n> FUNCtion | MATH} for DSO

		<source> ::= {CHANnel<n> DIGital0,..,DIGital15 | F | MATH} for MSO models <n> ::= 1-2 or 1-4 in NR1 <return_value> ::= ratio positive pulse width to p NR3 format
:MEASure:FALLtime [<source>]	:MEASure:FALLtime? [<source>]	<source> ::= {CHANnel<n> FUNCtion | MATH} for DSO <source> ::= {CHANnel<n> DIGital0,..,DIGital15 | F | MATH} for MSO models <n> ::= 1-2 or 1-4 in NR1 <return_value> ::= time i seconds between the lower upper thresholds in NR3 f
:MEASure:FREQuency [<source>]	:MEASure:FREQuency? [<source>]	<source> ::= {CHANnel<n> FUNCtion | MATH} for DSO <source> ::= {CHANnel<n> DIGital0,..,DIGital15 | F | MATH} for MSO models <n> ::= 1-2 or 1-4 in NR1 <return_value> ::= freque Hertz in NR3 format
:MEASure:NWIDth [<source>]	:MEASure:NWIDth? [<source>]	<source> ::= {CHANnel<n> FUNCtion | MATH} for DSO <source> ::= {CHANnel<n> DIGital0,..,DIGital15 | F | MATH} for MSO models <n> ::= 1-2 or 1-4 in NR1 <return_value> ::= negati width in seconds-NR3 form
:MEASure:OVERshoot [<source>]	:MEASure:OVERshoot? [<source>]	<source> ::= {CHANnel<n> FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 <return_value> ::= the pe the overshoot of the sele waveform in NR3 format
:MEASure:PERiod [<source>]	:MEASure:PERiod? [<source>]	<source> ::= {CHANnel<n> FUNCtion | MATH} for DSO <source> ::= {CHANnel<n> DIGital0,..,DIGital15 | F

		| MATH} for MSO models <n> ::= 1-2 or 1-4 in NR1 <return_value> ::= wavefo period in seconds in NR3
:MEASure:PHASe [<source1>] [,<source2>]	:MEASure:PHASe? [<source1>] [,<source2>]	<source1,2> ::= {CHANnel< FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 <return_value> ::= the ph angle value in degrees in format
:MEASure:PREShoot [<source>]	:MEASure:PREShoot? [<source>]	<source> ::= {CHANnel<n> FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 <return_value> ::= the pe preshoot of the selected in NR3 format
:MEASure:PWIDth [<source>]	:MEASure:PWIDth? [<source>]	<source> ::= {CHANnel<n> FUNCtion | MATH} for DSO <source> ::= {CHANnel<n> DIGital0,..,DIGital15 | F | MATH} for MSO models <n> ::= 1-2 or 1-4 in NR1 <return_value> ::= width positive pulse in seconds format
:MEASure:RISEtime [<source>]	:MEASure:RISEtime? [<source>]	<source> ::= {CHANnel<n> FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 <return_value> ::= rise t seconds in NR3 format
:MEASure:SDEViation [<source>]	:MEASure:SDEViation? [<source>]	<source> ::= {CHANnel<n> FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 <return_value> ::= calcul deviation in NR3 format
:MEASure:SHOW {1 | ON}	:MEASure:SHOW?	{1}
:MEASure:SOURce [<source1>] [,<source2>]	:MEASure:SOURce?	<source1,2> ::= {CHANnel< FUNCtion | MATH} for DSO <source1,2> ::= {CHANnel< DIGital0,..,DIGital15 | F

		| MATH} for MSO models <n> ::= 1-2 or 1-4 in NR1 <return_value> ::= {<sour NONE}
n/a	:MEASure:TEDGe? <slope><occurrence>[,<source>]	<slope> ::= direction of waveform <occurrence> ::= the tran to be reported <source> ::= {CHANnel<n> FUNCtion | MATH} for DSO <source> ::= {CHANnel<n> DIGital0,..,DIGital15 | F | MATH} for MSO models <n> ::= 1-2 or 1-4 in NR1 <return_value> ::= time i seconds of the specified transition
n/a	:MEASure:TVALue? <value>, [<slope>]<occurrence> [,<source>]	<value> ::= voltage level the waveform must cross. <slope> ::= direction of waveform when <value> is <occurrence> ::= transiti reported. <return_value> ::= time i seconds of specified volt crossing in NR3 format <source> ::= {CHANnel<n> FUNCtion | MATH} for DSO <source> ::= {CHANnel<n> DIGital0,..,DIGital15 | F | MATH} for MSO models <n> ::= 1-2 or 1-4 in NR1
:MEASure:VAMPlitude [<source>]	:MEASure:VAMPlitude? [<source>]	<source> ::= {CHANnel<n> FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 <return_value> ::= the am of the selected waveform in NR3 format
:MEASure:VAVerage [<source>]	:MEASure:VAVerage? [<source>]	<source> ::= {CHANnel<n> FUNCtion | MATH}

		<n> ::= 1-2 or 1-4 in NR1 <return_value> ::= calcul average voltage in NR3 fo
:MEASure:VBASe [<source>]	:MEASure:VBASe? [<source>]	<source> ::= {CHANnel<n> FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 <base_voltage> ::= voltag base of the selected wave NR3 format
:MEASure:VMAX [<source>]	:MEASure:VMAX? [<source>]	<source> ::= {CHANnel<n> FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 <return_value> ::= maximu voltage of the selected w in NR3 format
:MEASure:VMIN [<source>]	:MEASure:VMIN? [<source>]	<source> ::= {CHANnel<n> FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 <return_value> ::= minimu voltage of the selected w in NR3 format
:MEASure:VPP [<source>]	:MEASure:VPP? [<source>]	<source> ::= {CHANnel<n> FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 <return_value> ::= voltag to-peak of the selected w in NR3 format
:MEASure:VRMS [<source>]	:MEASure:VRMS? [<source>]	<source> ::= {CHANnel<n> FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 <return_value> ::= calcul RMS voltage in NR3 format
n/a	:MEASure:VTIMe? <vtime> [,<source>]	<vtime> ::= displayed tim trigger in seconds in NR3 <return_value> ::= voltag specified time in NR3 for <source> ::= {CHANnel<n> FUNCtion | MATH} for DSO <source> ::= {CHANnel<n> DIGital0,..,DIGital15 | F | MATH} for MSO models

		<n> ::= 1-2 or 1-4 in NR1
:MEASure:VTOP [<source>]	:MEASure:VTOP? [<source>]	<source> ::= {CHANnel<n> FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 <return_value> ::= voltag top of the waveform in NR
:MEASure:XMAX [<source>]	:MEASure:XMAX? [<source>]	<source> ::= {CHANnel<n> FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 <return_value> ::= horizo value of the maximum in N format
:MEASure:XMIN [<source>]	:MEASure:XMIN? [<source>]	<source> ::= {CHANnel<n> FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 <return_value> ::= horizo value of the maximum in N format
:MERGe <pixel memory>	n/a	<pixel memory> ::= {PMEMo | 2 | 3 | 4 | 5 | 6 | 7 |
*OPC	*OPC?	ASCII "1" is placed in th queue when all pending de operations have completed
:OPEE <n>	:OPEE?	<n> ::= 16-bit integer in format
n/a	:OPERegister:CONDition?	<n> ::= 16-bit integer in format
n/a	:OPERegister[:EVENt]?	<n> ::= 16-bit integer in format
n/a	*OPT?	<return_value> ::= 0,0,<l info> <license info> ::= <All f <reserved>, <Factory MSO> <Upgraded MSO>, <Probe fi <Memory>, <Low Speed Seri <reserved>, <reserved> <All field> ::= {0 | All} <reserved> ::= 0 <Factory MSO> ::= {0 | MS <Upgraded MSO> ::= {0 | M <Probe field> ::= 0

<table>
<tr><td></td><td></td><td>&lt;Memory> ::= {0 | mem2M |

<Low Speed Serial> ::= {0

<reserved> ::= 0

<reserved&gt; ::= 0</td></tr>
<tr><td>:OVLenable <mask&gt;</td><td>:OVLenable?</td><td>&lt;mask&gt; ::= 16-bit integer format as shown:

<table>
<tr><th>Bit</th><th>Weight</th><th>Input</th></tr>
<tr><td>10</td><td>1024</td><td>External Tri Fault</td></tr>
<tr><td>9</td><td>512</td><td>Channel 4 Fau</td></tr>
<tr><td>8</td><td>256</td><td>Channel 3 Fau</td></tr>
<tr><td>7</td><td>128</td><td>Channel 2 Fau</td></tr>
<tr><td>6</td><td>64</td><td>Channel 1 Fau</td></tr>
<tr><td>4</td><td>16</td><td>External Tri OVL</td></tr>
<tr><td>3</td><td>8</td><td>Channel 4 OV</td></tr>
<tr><td>2</td><td>4</td><td>Channel 3 OV</td></tr>
<tr><td>1</td><td>2</td><td>Channel 2 OV</td></tr>
<tr><td>0</td><td>1</td><td>Channel 1 OV</td></tr>
</table></td></tr>
<tr><td>n/a</td><td>:OVLRegister?</td><td>&lt;value> ::= integer in NR format. See OVLenable for</td></tr>
<tr><td>:POD&lt;n>:DISPlay {{0 | OFF} | {1 | ON}}</td><td>:POD&lt;n>:DISPlay?</td><td>{0 | 1}

<n> ::= 1-2 in NR1 format</td></tr>
<tr><td>:POD&lt;n>:SIZE <value&gt;</td><td>:POD&lt;n>:SIZE?</td><td>&lt;value> ::= {SMALl | MEDi LARGe}</td></tr>
<tr><td>:POD&lt;n>:THReshold <type>[suffix]</td><td>:POD&lt;n>:THReshold?</td><td>&lt;n&gt; ::= 1-2 in NR1 format

<type> ::= {CMOS | ECL | <user defined value>}

<user defined value> ::= NR3 format

[suffix] ::= {V | mV | uV</td></tr>
<tr><td>:PRINt [<options&gt;]</td><td>n/a</td><td>&lt;options> ::= [<print opt [,..,<print option>]

<print option> ::= {COLor GRAYscale | PRINter0 | BM BMP | PNG | NOFactors | F

<print option> can be rep</td></tr>
</table>

		to 5 times.
*RCL <value>	n/a	<value> ::= {0 | 1 | 2 | 5 | 6 | 7 | 8 | 9}
*RST	n/a	See *RST (Reset)
:RUN	n/a	n/a
*SAV <value>	n/a	<value> ::= {0 | 1 | 2 | 5 | 6 | 7 | 8 | 9}
n/a	:SBUS:CAN:COUNt:ERRor?	<frame_count> ::= integer format
n/a	:SBUS:CAN:COUNt:OVERload?	<frame_count> ::= integer format
:SBUS:CAN:COUNt:RESet	n/a	n/a
n/a	:SBUS:CAN:COUNt:TOTal?	<frame_count> ::= integer format
n/a	:SBUS:CAN:COUNt:UTILization?	<percent> ::= floating-po NR3 format
:SBUS:DISPlay {{0 | OFF} | {1 | ON}}	:SBUS:DISPlay?	{0 | 1}
:SBUS:IIC:ASIZe <size>	:SBUS:IIC:ASIZe?	<size> ::= {BIT7 | BIT8}
:SBUS:LIN:PARity {{0 | OFF} | {1 | ON}}	:SBUS:LIN:PARity?	{0 | 1}
:SBUS:MODE <mode>	:SBUS:MODE?	<mode> ::= {IIC | SPI | C LIN}
:SBUS:SPI:WIDTh <word_width>	:SBUS:SPI:WIDTh?	<word_width> ::= integer NR1 format
n/a	:SERial	<return value> ::= unquot string containing serial
:SINGle	n/a	n/a
*SRE <mask>	*SRE?	<mask> ::= sum of all bit are set, 0 to 255; an int NR1 format. <mask> ::= fo values:

Bit	Weight	Name	Enables
7	128	OPER	Operati Status Registe
6	64	----	(Not us
5	32	ESB	Event S Bit
4	16	MAV	Message Availab
3	8	----	(Not us
2	4	MSG	Message

<table>
<tr><td></td><td></td><td><table><tr><td>1</td><td>2</td><td>USR</td><td>User</td></tr><tr><td>0</td><td>1</td><td>TRG</td><td>Trigger</td></tr></table></td></tr>
<tr><td>n/a</td><td>:STATus? <display></td><td>{0 | 1}
<display> ::= {CHANnel<n> DIGital0,..,DIGital15 | P 2} | BUS{1 | 2} | FUNCtio | SBUS}
<n> ::= 1-2 or 1-4 in NR1</td></tr>
<tr><td>n/a</td><td>*STB?</td><td><value> ::= 0 to 255; an in NR1 format, as shown i following:<table><tr><th>Bit</th><th>Weight</th><th>Name</th><th>"1" Ind</th></tr><tr><td>7</td><td>128</td><td>OPER</td><td>An enab operati status conditi occurre</td></tr><tr><td>6</td><td>64</td><td>RQS/ MSS</td><td>Instrum request service</td></tr><tr><td>5</td><td>32</td><td>ESB</td><td>Enabled status conditi occurre</td></tr><tr><td>4</td><td>16</td><td>MAV</td><td>An outp message ready.</td></tr><tr><td>3</td><td>8</td><td>----</td><td>(Not us</td></tr><tr><td>2</td><td>4</td><td>MSG</td><td>Message been display</td></tr><tr><td>1</td><td>2</td><td>USR</td><td>An enab user ev conditi occurre</td></tr><tr><td>0</td><td>1</td><td>TRG</td><td>A trigg occurre</td></tr></table></td></tr>
<tr><td>:STOP</td><td>n/a</td><td>n/a</td></tr>
<tr><td>:SYSTem:DATE <date></td><td>:SYSTem:DATE?</td><td><date> ::= <year>,<month>
<year> ::= 4-digit year i format
<month> ::= {1,..,12 | JA FEBruary | MARch | APRil JUNe | JULy | AUGust | SE | OCTober | NOVember | DE</td></tr>
</table>

		<day> ::= {1,..31}
:SYSTem:DSP <string>	n/a	<string> ::= up to 254 ch as a quoted ASCII string
n/a	:SYSTem:ERRor?	<error> ::= an integer er <error string> ::= quoted string. See Error Messages.
:SYSTem:LOCK	:SYSTem:LOCK?	<value> ::= {ON | OFF}
:SYSTem:SETup <setup_data>	:SYSTem:SETup?	<setup_data> ::= data in 488.2 # format.
:SYSTem:TIME <time>	:SYSTem:TIME?	<time> ::= hours,minutes, in NR1 format
n/a	:TER?	{0 | 1}
:TIMebase:MODE <value>	:TIMebase:MODE?	<value> ::= {MAIN | WINDo ROLL}
:TIMebase:POSition <pos>	:TIMebase:POSition?	<pos> ::= time from the t event to the display refe point in NR3 format
:TIMebase:RANGe <range_value>	:TIMebase:RANGe?	<range_value> ::= 5 ns th 500 s in NR3 format
:TIMebase:REFClock {{0 | OFF} | {1 | ON}}	:TIMebase:REFClock?	{0 | 1}
:TIMebase:REFerence {LEFT | CENTer | RIGHt}	:TIMebase:REFerence?	<return_value> ::= {LEFT | RIGHt}
:TIMebase:SCALe <scale_value>	:TIMebase:SCALe?	<scale_value> ::= scale v seconds in NR3 format
:TIMebase:VERNier {{0 | OFF} | {1 | ON}}	:TIMebase:VERNier?	{0 | 1}
:TIMebase:WINDow:POSition <pos>	:TIMebase:WINDow:POSition?	<pos> ::= time from the t event to the delayed view reference point in NR3 fo
:TIMebase:WINDow:RANGe <range_value>	:TIMebase:WINDow:RANGe?	<range value> ::= range v seconds in NR3 format for delayed window
:TIMebase:WINDow:SCALe <scale_value>	:TIMebase:WINDow:SCALe?	<scale_value> ::= scale v seconds in NR3 format for delayed window
*TRG	n/a	n/a
:TRIGger:CAN:PATTern:DATA <value>, <mask>	:TRIGger:CAN:PATTern:DATA?	<value> ::= 64-bit intege decimal, <nondecimal>, or <string> (with Option AMS <mask> ::= 64-bit integer decimal, <nondecimal>, or <string>

		<nondecimal> ::= #Hnn...n n ::= {0,..,9 | A,..,F} f hexadecimal <nondecimal> ::= #Bnn...n n ::= {0 | 1} for binary <string> ::= "0xnn...n" w n ::= {0,..,9 | A,..,F} f hexadecimal
:TRIGger:CAN:PATTern:DATA:LENGth <length>	:TRIGger:CAN:PATTern:DATA:LENGth?	<length> ::= integer from in NR1 format (with Optio
:TRIGger:CAN:PATTern:ID <value>, <mask>	:TRIGger:CAN:PATTern:ID?	<value> ::= 32-bit intege decimal, <nondecimal>, or <string> (with Option AMS <mask> ::= 32-bit integer decimal, <nondecimal>, or <string> <nondecimal> ::= #Hnn...n n ::= {0,..,9 | A,..,F} f hexadecimal <nondecimal> ::= #Bnn...n n ::= {0 | 1} for binary <string> ::= "0xnn...n" w n ::= {0,..,9 | A,..,F} f hexadecimal
:TRIGger:CAN:PATTern:ID:MODE <value>	:TRIGger:CAN:PATTern:ID:MODE?	<value> ::= {STANdard | E (with Option AMS)
:TRIGger:CAN:SAMPlepoint <value>	:TRIGger:CAN:SAMPlepoint?	<value> ::= {60 | 62.5 | | 75 | 80 | 87.5} in NR3 (with Option AMS)
:TRIGger:CAN:SIGNal:BAUDrate <baudrate>	:TRIGger:CAN:SIGNal:BAUDrate?	<baudrate> ::= {10000 | 2 33300 | 50000 | 62500 | 8 100000 | 125000 | 250000 | 800000 | 1000000}
:TRIGger:CAN:SOURce <source>	:TRIGger:CAN:SOURce?	<source> ::= {CHANnel<n> EXTernal} for DSO models <source> ::= {CHANnel<n> DIGital0,..,DIGital15 |} models <n> ::= 1-2 or 1-4 in NR1
:TRIGger:CAN:TRIGger <condition>	:TRIGger:CAN:TRIGger?	<condition> ::= {SOF} (wi Option AMS) <condition> ::= {SOF | DA ERRor | IDData | IDEither IDRemote | ALLerrors | OV ACKerror} (with Option AM
:TRIGger:DURation:GREaterthan	:TRIGger:DURation:GREaterthan?	<greater than time> ::= f

<greater than time>[suffix]		point number from 5 ns to 10 seconds in NR3 format [suffix] ::= {s | ms | us ps}
:TRIGger:DURation:LESSthan <less than time>[suffix]	:TRIGger:DURation:LESSthan?	<less than time> ::= floa point number from 5 ns to 10 seconds in NR3 format [suffix] ::= {s | ms | us ps}
:TRIGger:DURation:PATTern <value>, <mask>	:TRIGger:DURation:PATTern?	<value> ::= integer or <s <mask> ::= integer or <st <string> ::= ""0xnnnnnn"" {0,..,9 | A,..,F}
:TRIGger:DURation:QUALifier <qualifier>	:TRIGger:DURation:QUALifier?	<qualifier> ::= {GREatert LESSthan | INRange | OUTR TIMeout}
:TRIGger:DURation:RANGe <greater than time>[suffix], <less than time>[suffix]	:TRIGger:DURation:RANGe?	<greater than time> ::= m duration from 10 ns to 9.99 seconds in NR3 forma <less than time> ::= max from 15 ns to 10 seconds format [suffix] ::= {s | ms | us ps}
:TRIGger:EBURst:COUNt <count>	:TRIGger:EBURst:COUNt?	<count> ::= integer in NR
:TRIGger:EBURst:IDLE <time_value>	:TRIGger:EBURst:IDLE?	<time_value> ::= time in in NR3 format
:TRIGger:EBURst:SLOPe <slope>	:TRIGger:EBURst:SLOPe?	<slope> ::= {NEGative | P
:TRIGger[:EDGE]:COUPling {AC | DC | LF}	:TRIGger[:EDGE]:COUPling?	{AC | DC | LF}
:TRIGger[:EDGE]:LEVel <level> [,<source>]	:TRIGger[:EDGE]:LEVel? [<source>]	For internal triggers, <level> ::= .75 x full-sc voltage from center scree format. For external triggers, <level> ::= 2 volts with attenuation at 1:1 in NR3 For digital channels (MSO models), <level> ::= 8 V. <source> ::= {CHANnel<n> EXTernal} for DSO models <source> ::= {CHANnel<n> DIGital0,..,DIGital15 | EXTernal } for MSO models

		<n> ::= 1-2 or 1-4 in NR1
:TRIGger[:EDGE]:REJect {OFF | LF | HF}	:TRIGger[:EDGE]:REJect?	{OFF | LF | HF}
:TRIGger[:EDGE]:SLOPe <polarity>	:TRIGger[:EDGE]:SLOPe?	<polarity> ::= {POSitive NEGative | EITHer | ALTer
:TRIGger[:EDGE]:SOURce <source>	:TRIGger[:EDGE]:SOURce?	<source> ::= {CHANnel<n> EXTernal} for DSO models <source> ::= {CHANnel<n> DIGital0,..,DIGital15 | E for MSO models <n> ::= 1-2 or 1-4 in NR1
:TRIGger:GLITch:GREaterthan <greater than time>[suffix]	:TRIGger:GLITch:GREaterthan?	<greater than time> ::= f point number from 5 ns to 10 seconds in NR3 format [suffix] ::= {s | ms | us ps}
:TRIGger:GLITch:LESSthan <less than time>[suffix]	:TRIGger:GLITch:LESSthan?	<less than time> ::= floa point number from 5 ns to 10 seconds in NR3 format [suffix] ::= {s | ms | us ps}
:TRIGger:GLITch:LEVel <level> [<source>]	:TRIGger:GLITch:LEVel?	For internal triggers, <level> ::= .75 x full-sc voltage from center scree format. For external triggers, <level> ::= 2 volts with attenuation at 1:1 in NR3 For digital channels (MSO models), <level> ::= 6 V. <source> ::= {CHANnel<n> EXTernal} for DSO models <source> ::= {CHANnel<n> DIGital0,..,DIGital15} fo models <n> ::= 1-2 or 1-4 in NR1
:TRIGger:GLITch:POLarity <polarity>	:TRIGger:GLITch:POLarity?	<polarity> ::= {POSitive NEGative}
:TRIGger:GLITch:QUALifier <qualifier>	:TRIGger:GLITch:QUALifier?	<qualifier> ::= {GREatert LESSthan | RANGe}
:TRIGger:GLITch:RANGe <greater than time>[suffix], <less than time>[suffix]	:TRIGger:GLITch:RANGe?	<greater than time> ::= s time from 10 ns to 9.99 s in NR3 format <less than time> ::= stop

		from 15 ns to 10 seconds format
		[suffix] ::= {s | ms | us ps}

:TRIGger:GLITch:SOURce <source>	:TRIGger:GLITch:SOURce?	<source> ::= {CHANnel<n> EXTernal} for DSO models <source> ::= {CHANnel<n> DIGital0,..,DIGital15 } f models <n> ::= 1-2 or 1-4 in NR1
:TRIGger:HFReject {{0 | OFF} | {1 | ON}}	:TRIGger:HFReject?	{0 | 1}
:TRIGger:HOLDoff <holdoff_time>	:TRIGger:HOLDoff?	<holdoff_time> ::= 60 ns in NR3 format
:TRIGger:IIC:PATTern:ADDRess <value>	:TRIGger:IIC:PATTern:ADDRess?	<value> ::= integer or <s <string> ::= "0xnn" n ::= | A,..,F}
:TRIGger:IIC:PATTern:DATA <value>	:TRIGger:IIC:PATTern:DATA?	<value> ::= integer or <s <string> ::= "0xnn" n ::= | A,..,F}
:TRIGger:IIC:PATTern:DATa2 <value>	:TRIGger:IIC:PATTern:DATa2?	<value> ::= integer or <s <string> ::= "0xnn" n ::= | A,..,F}
:TRIGger:IIC[:SOURce]:CLOCk <source>	:TRIGger:IIC[:SOURce]:CLOCk?	<source> ::= {CHANnel<n> EXTernal} for DSO models <source> ::= {CHANnel<n> DIGital0,..,DIGital15 } f models <n> ::= 1-2 or 1-4 in NR1
:TRIGger:IIC[:SOURce]:DATA <source>	:TRIGger:IIC[:SOURce]:DATA?	<source> ::= {CHANnel<n> EXTernal} for DSO models <source> ::= {CHANnel<n> DIGital0,..,DIGital15 } f models <n> ::= 1-2 or 1-4 in NR1
:TRIGger:IIC:TRIGger:QUALifier <value>	:TRIGger:IIC:TRIGger:QUALifier?	<value> ::= {EQUal | NOTe LESSthan | GREaterthan}
:TRIGger:IIC:TRIGger[:TYPE] <type>	:TRIGger:IIC:TRIGger[:TYPE]?	<type> ::= {STARt | STOP | READEeprom | WRITe7 | W NACKnowledge | ANACknowle R7Data2 | W7Data2 | RESTa
:TRIGger:LIN:ID <value>	:TRIGger:LIN:ID?	<value> ::= 7-bit integer

		decimal, <nondecimal>, or <string> from 0-63 or 0x0 (with Option AMS) <nondecimal> ::= #Hnn whe {0,..,9 | A,..,F} for hex <nondecimal> ::= #Bnn...n n ::= {0 | 1} for binary <string> ::= "0xnn" where {0,..,9 | A,..,F} for hex
:TRIGger:LIN:SAMPlepoint <value>	:TRIGger:LIN:SAMPlepoint?	<value> ::= {60 | 62.5 | | 75 | 80 | 87.5} in NR3
:TRIGger:LIN:SIGNal:BAUDrate <baudrate>	:TRIGger:LIN:SIGNal:BAUDrate?	<baudrate> ::= {2400 | 96 19200}
:TRIGger:LIN:SOURce <source>	:TRIGger:LIN:SOURce?	<source> ::= {CHANnel<n> EXTernal} for DSO models <source> ::= {CHANnel<n> DIGital0,..,DIGital15} fo models <n> ::= 1-2 or 1-4 in NR1
:TRIGger:LIN:STANdard <std>	:TRIGger:LIN:STANdard?	<std> ::= {LIN13 | LIN20}
:TRIGger:LIN:SYNCbreak <value>	:TRIGger:LIN:SYNCbreak?	<value> ::= integer = {11 13}
:TRIGger:LIN:TRIGger <condition>	:TRIGger:LIN:TRIGger?	<condition> ::= {SYNCbrea (without Option AMS) <condition> ::= {SYNCbrea (with Option AMS)

:TRIGger:MODE <mode>	:TRIGger:MODE?	<mode> ::= {EDGE | GLITch PATTern | CAN | DURation EBURst | LIN | SEQuence | TV | USB}
		<return_value> ::= {<mode <none>}
		<none> ::= query returns if the :TIMebase:MODE is XY

:TRIGger:NREJect {{0 | OFF} | {1 | ON}}	:TRIGger:NREJect?	{0 | 1}
:TRIGger:PATTern <value>, <mask> [,<edge source>,<edge>]	:TRIGger:PATTern?	<value> ::= 32-bit intege <string> <mask> ::= 32-bit integer <string> <string> ::= "0xnnnnnn"; {0,..,9 | A,..,F}

		<edge source> ::= {CHANNe EXTernal | NONE} for DSO <edge source> ::= {CHANNe DIGital0,..,DIGital15 | N MSO models <edge> ::= {POSitive | NE <n> ::= 1-2 or 1-4 in NR1
:TRIGger:SEQuence:COUNt <count>	:TRIGger:SEQuence:COUNt?	<count> ::= integer in NR
:TRIGger:SEQuence:EDGE{1|2} <source>, <slope>	:TRIGger:SEQuence:EDGE{1|2}?	<source> ::= {CHANnel<n> EXTernal} for the DSO mod <source> ::= {CHANnel<n> DIGital0,..,DIGital15} fo MSO models <slope> ::= {POSitive | N <n> ::= 1-2 or 1-4 in NR1 <return_value> ::= query "NONE" if edge source is
:TRIGger:SEQuence:FIND <value>	:TRIGger:SEQuence:FIND?	<value> ::= {PATTern1,ENT PATTern1,EXITed | EDGE1 | PATTern1,AND,EDGE1}
:TRIGger:SEQuence:PATTern{1|2} <value>, <mask>	:TRIGger:SEQuence:PATTern{1|2}?	<value> ::= integer or <s <mask> ::= integer or <st <string> ::= "0xnnnnnn" n {0,..,9 | A,..,F}
:TRIGger:SEQuence:RESet <value>	:TRIGger:SEQuence:RESet?	<value> ::= {NONE | PATTern1,ENTered | PATTern1,EXITed | EDGE1 | PATTern1,AND,EDGE1 | PATTern2,ENTered | PATTern2,EXITed | EDGE2 | Values used in find and t stages not available. EDG available if EDGE2,COUNt trigger stage.
:TRIGger:SEQuence:TIMer <time_value>	:TRIGger:SEQuence:TIMer?	<time_value> ::= time fro to 10 seconds in NR3 form
:TRIGger:SEQuence:TRIGger <value>	:TRIGger:SEQuence:TRIGger?	<value> ::= {PATTern2,ENT PATTern2,EXITed | EDGE2 | PATTern2,AND,EDGE2 | EDGE | EDGE2,COUNt,NREFind}
:TRIGger:SPI:CLOCk:SLOPe <slope>	:TRIGger:SPI:CLOCk:SLOPe?	<slope> ::= {NEGative | P
:TRIGger:SPI:CLOCk:TIMeout <time_value>	:TRIGger:SPI:CLOCk:TIMeout?	<time_value> ::= time in in NR1 format

:TRIGger:SPI:FRAMing <value>	:TRIGger:SPI:FRAMing?	<value> ::= {CHIPselect | NOTChipselect | TIMeout}
:TRIGger:SPI:PATTern:DATA <value>, <mask>	:TRIGger:SPI:PATTern:DATA?	<value> ::= integer or <s <mask> ::= integer or <st <string> ::= "0xnnnnnn" w n ::= {0,..,9 | A,..,F}
:TRIGger:SPI:PATTern:WIDTh <width>	:TRIGger:SPI:PATTern:WIDTh?	<width> ::= integer from in NR1 format
:TRIGger:SPI:SOURce:CLOCk <source>	:TRIGger:SPI:SOURce:CLOCk?	<value> ::= {CHANnel<n> | EXTernal} for the DSO mod <value> ::= {CHANnel<n> | DIGital0,..,DIGital15} fo MSO models <n> ::= 1-2 or 1-4 in NR1
:TRIGger:SPI:SOURce:DATA <source>	:TRIGger:SPI:SOURce:DATA?	<value> ::= {CHANnel<n> | EXTernal} for the DSO mod <value> ::= {CHANnel<n> | DIGital0,..,DIGital15} fo MSO models <n> ::= 1-2 or 1-4 in NR1
:TRIGger:SPI:SOURce:FRAMe <source>	:TRIGger:SPI:SOURce:FRAMe?	<value> ::= {CHANnel<n> | EXTernal} for the DSO mod <value> ::= {CHANnel<n> | DIGital0,..,DIGital15} fo MSO models <n> ::= 1-2 or 1-4 in NR1
:TRIGger:SWEep <sweep>	:TRIGger:SWEep?	<sweep> ::= {AUTO | NORMa
:TRIGger:TV:LINE <line number>	:TRIGger:TV:LINE?	<line number> ::= integer format
:TRIGger:TV:MODE <tv mode>	:TRIGger:TV:MODE?	<tv mode> ::= {FIEld1 | F AFIelds | ALINes | LINE | VERTical | LFIeld1 | LFIe LALTernate | LVERtical}
:TRIGger:TV:POLarity <polarity>	:TRIGger:TV:POLarity?	<polarity> ::= {POSitive NEGative}
:TRIGger:TV:SOURce <source>	:TRIGger:TV:SOURce?	<source> ::= {CHANnel<n>} <n> ::= 1-2 or 1-4 intege format
:TRIGger:TV:STANdard <standard>	:TRIGger:TV:STANdard?	<standard> ::= {GENeric | PALM | PAL | SECam | {P48 P480} | {P720L60HZ | P720 {P1080L24HZ | P1080} | P1 | {I1080L50HZ | I1080} |

		I1080L60HZ}
:TRIGger:USB:SOURce:DMINus <source>	:TRIGger:USB:SOURce:DMINus?	<source> ::= {CHANnel<n> EXTernal} for the DSO mod <source> ::= {CHANnel<n> DIGital0,..,DIGital15} fo MSO models <n> ::= 1-2 or 1-4 in NR1
:TRIGger:USB:SOURce:DPLus <source>	:TRIGger:USB:SOURce:DPLus?	<source> ::= {CHANnel<n> EXTernal} for the DSO mod <source> ::= {CHANnel<n> DIGital0,..,DIGital15} fo MSO models <n> ::= 1-2 or 1-4 in NR1
:TRIGger:USB:SPEed <value>	:TRIGger:USB:SPEed?	<value> ::= {LOW | FULL}
:TRIGger:USB:TRIGger <value>	:TRIGger:USB:TRIGger?	<value> ::= {SOP | EOP | ENTersuspend | EXITsuspen RESet}
n/a	*TST?	<result> ::= 0 or non-zer an integer in NR1 format
:VIEW <source>	n/a	<source> ::= {CHANnel<n> PMEMory{0 | 1 | 2 | 3 | 4 | 7 | 8 | 9} | FUNCtion | SBUS} for DSO models <source> ::= {CHANnel<n> DIGital0,..,DIGital15 | P | 1 | 2 | 3 | 4 | 5 | 6 | 9} | POD{1 | 2} | BUS{1 | FUNCtion | MATH | SBUS} f models <n> ::= 1-2 or 1-4 in NR1
*WAI	n/a	n/a
:WAVeform:BYTeorder <value>	:WAVeform:BYTeorder?	<value> ::= {LSBFirst | M
n/a	:WAVeform:COUNt?	<count> ::= an integer fr 65536 in NR1 format
n/a	:WAVeform:DATA?	<binary block length byte <binary data> For example, to transmit bytes of data, the syntax be: #800001000<1000 bytes data><NL> 8 is the number of digits follow 00001000 is the number of to be transmitted

		<1000 bytes of data> is t actual data
:WAVeform:FORMat <value>	:WAVeform:FORMat?	<value> ::= {WORD | BYTE
:WAVeform:POINts <# points>	:WAVeform:POINts?	<# points> ::= {100 | 250 1000 | <points_mode>} if points mode is NORMal <# points> ::= {100 | 250 1000 | 2000 ... 8000000 i sequence | <points_mode>} waveform points mode is M or RAW <points_mode> ::= {NORMal MAXimum | RAW}
:WAVeform:POINts:MODE <points_mode>	:WAVeform:POINts:MODE?	<points_mode> ::= {NORMal MAXimum | RAW}
n/a	:WAVeform:PREamble?	<preamble_block> ::= <for NR1>, <type NR1>,<points NR1>,<count NR1>, <xincre NR3>, <xorigin NR3>, <xre NR1>,<yincrement NR3>, <y NR3>, <yreference NR1> <format> ::= an integer i format: • 0 for BYTE format • 1 for WORD format • 2 for ASCii format <type> ::= an integer in format: • 0 for NORMal type • 1 for PEAK detect typ • 2 for AVERage type • 3 for HRESolution typ <count> ::= Average count if PEAK detect type or NO integer in NR1 format
:WAVeform:SOURce <source>	:WAVeform:SOURce?	<source> ::= {CHANnel<n> FUNCtion | MATH | SBUS} f models <source> ::= {CHANnel<n> | 2} | BUS{1 | 2} | FUNCt MATH | SBUS} for MSO mode <n> ::= 1-2 or 1-4 in NR1

n/a	:WAVeform:TYPE?	<return_mode> ::= {NORM | AVER | HRES}
:WAVeform:UNSigned {{0 | OFF} | {1 | ON}}	:WAVeform:UNSigned?	{0 | 1}
:WAVeform:VIEW <view>	:WAVeform:VIEW?	<view> ::= {MAIN}
n/a	:WAVeform:XINCrement?	<return_value> ::= x-incr the current preamble in N format
n/a	:WAVeform:XORigin?	<return_value> ::= x-orig in the current preamble i format
n/a	:WAVeform:XREFerence?	<return_value> ::= 0 (x-r value in the current prea NR1 format)
n/a	:WAVeform:YINCrement?	<return_value> ::= y-incr value in the current prea NR3 format
n/a	:WAVeform:YORigin?	<return_value> ::= y-orig the current preamble in N format
n/a	:WAVeform:YREFerence?	<return_value> ::= y-refe value in the current prea NR1 format

Agilent 6000 Series Oscilloscopes Programmer's Reference

Commands by Subsystem

Subsystem	Description
:ACQuire Commands	Set the parameters for acquiring and storing data.
:BUS<n> Commands	Control all oscilloscope functions associated with the digital channels bus display.
:CALibrate Commands	Utility commands for determining the state of the calibration factor protection switch.
:CHANnel<n> Commands	Control all oscilloscope functions associated with individual analog channels or groups of channels.
Common (*) Commands	Commands defined by IEEE 488.2 standard that are common to all instruments.
:DIGital<n> Commands	Control all oscilloscope functions associated with individual digital channels.
:DISPlay Commands	Control how waveforms, graticule, and text are displayed and written on the screen.
:EXTernal Trigger Commands	Control the input characteristics of the external trigger input.
:FUNCtion Commands	Control functions in the measurement/storage module.
:HARDcopy Commands	Set and query the selection of hardcopy device and formatting options.
:MARKer Commands	Set and query the settings of X-axis markers (X1 and X2 cursors) and the Y-axis markers (Y1 and Y2 cursors).

:MEASure Commands	Select automatic measurements to be made and control time markers.
:POD Commands	Control all oscilloscope functions associated with groups of digital channels.
Root (:) Commands	Control many of the basic functions of the oscilloscope and reside at the root level of the command tree.
:SBUS Commands	Control oscilloscope functions associated with the serial decode bus.
:SYSTem Commands	Control basic system functions of the oscilloscope.
:TIMebase Commands	Control all horizontal sweep functions.
:TRIGger Commands	Control the trigger modes and parameters for each trigger type.
:WAVeform Commands	Provide access to waveform data.

Command Types

Three types of commands are used:

- **Common (*) Commands** — See Introduction to Common (*) Commands for more information.
- **Root Level (:) Commands** — See Introduction to Root (:) Commands for more information.
- **Subsystem Commands** — Subsystem commands are grouped together under a common node of the Command Tree, such as the :TIMebase commands. Only one subsystem may be selected at any given time. When the instrument is initially turned on, the command parser is set to the root of the command tree; therefore, no subsystem is selected.

Commands by Subsystem

:ACQuire Commands

Set the parameters for acquiring and storing data. See Introduction to :ACQuire Commands.

Command	Query	Options and Query Returns
n/a	:ACQuire:AALias?	{1 | 0}
:ACQuire:COMPlete <complete>	:ACQuire:COMPlete?	<complete> ::= 100; an integer in NR1 format
:ACQuire:COUNt <count>	:ACQuire:COUNt?	<count> ::= an integer from 1 to 65536 in NR1 format
:ACQuire:DAALias <mode>	:ACQuire:DAALias?	<mode> ::= {DISable | AUTO}
:ACQuire:MODE <mode>	:ACQuire:MODE?	<mode> ::= {RTIMe | ETIMe}
n/a	:ACQuire:POINts?	<# points> ::= an integer in NR1 format
:ACQuire:RSIGnal <ref_signal_mode>	:ACQuire:RSIGnal?	<ref_signal_mode> ::= {OFF | OUT | IN}
n/a	:ACQuire:SRATe?	<sample_rate> ::= sample rate (samples/s) in NR3 format
:ACQuire:TYPE <type>	:ACQuire:TYPE?	<type> ::= {NORMal | AVERage | HRESolution | PEAK}

Introduction to :ACQuire Commands

The ACQuire subsystem controls the way in which waveforms are acquired. These acquisition types are available: normal, averaging, peak detect, and high resolution. Two acquisition modes are available: real-time mode, and equivalent-time mode.

Normal. The :ACQuire:TYPE NORMal command sets the oscilloscope in the normal acquisition mode. For the majority of user models and signals, NORMal mode yields the best oscilloscope picture of the waveform.

Averaging. The :ACQuire:TYPE AVERage command sets the oscilloscope in the averaging mode. You can set the count by sending the :ACQuire:COUNt command followed by the number of averages. In this mode, the value for averages is an integer from 1 (smoothing) to 65536. The COUNt value determines the number of averages that must be acquired.

Peak Detect. The :ACQuire:TYPE PEAK command sets the oscilloscope in the peak detect mode. In this mode, :ACQuire:COUNt has no meaning.

Real-time Mode. The :ACQuire:MODE RTIMe command sets the oscilloscope in real-time mode. This mode is useful to inhibit equivalent time sampling at fast sweep speeds.

Equivalent-time Mode. The :ACQuire:MODE ETIME command sets the oscilloscope in equivalent-time mode.

Reporting the Setup. Use :ACQuire? to query setup information for the ACQuire subsystem.

Return Format. The following is a sample response from the :ACQuire? query. In this case, the query was issued following a *RST command.

```
:ACQ:MODE RTIM;TYPE NORM;COMP 100;COUNT 8
```

:ACQuire Commands

:ACQuire:AALias — N

Query Syntax

```
:ACQuire:AALias?
```

The :ACQuire:AALias? query returns the current state of the oscilloscope acquisition anti-alias control. This control can be directly disabled or disabled automatically.

Return Format

```
<value><NL>

<value> ::= {1 | 0}
```

See Also

- Introduction to :ACQuire Commands
- :ACQuire:DAALias

:ACQuire Commands

:ACQuire:COMPlete — C

Command Syntax

```
:ACQuire:COMPlete <complete>
```

```
<complete> ::= 100; an integer in NR1 format
```

The :ACQuire:COMPlete command affects the operation of the :DIGitize command. It specifies the minimum completion criteria for an acquisition. The parameter determines the percentage of the time buckets that must be "full" before an acquisition is considered complete. If :ACQuire:TYPE is NORMal, it needs only one sample per time bucket for that time bucket to be considered full.

The only legal value for the :COMPlete command is 100. All time buckets must contain data for the acquisition to be considered complete.

Query Syntax

```
:ACQuire:COMPlete?
```

The :ACQuire:COMPlete? query returns the completion criteria (100) for the currently selected mode.

Return Format

```
<completion_criteria><NL>
```

```
<completion_criteria> ::= 100; an integer in NR1 format
```

See Also

- Introduction to :ACQuire Commands
- :DIGitize
- :WAVeform:POINts

Example Code

```
   ' AQUIRE_COMPLETE - Specifies the minimum completion criteria for
   ' an acquisition.  The parameter determines the percentage of time
   ' buckets needed to be "full" before an acquisition is considered
   ' to be complete.
   myScope.WriteString ":ACQUIRE:COMPLETE 100"
```

Example program from the start: VISA COM Example in Visual Basic

:ACQuire Commands

:ACQuire:COUNt — C

Command Syntax

```
:ACQuire:COUNt <count>
```

```
<count> ::= integer in NR1 format
```

In averaging mode, the :ACQuire:COUNt command specifies the number of values to be averaged for each time bucket before the acquisition is considered to be complete for that time bucket. When :ACQuire:TYPE is set to AVERage, the count can be set to any value from 1 (smoothing) to 65536.

Query Syntax

```
:ACQuire:COUNT?
```

The :ACQuire:COUNT? query returns the currently selected count value for averaging mode.

Return Format

```
<count_argument><NL>
```

```
<count_argument> ::= an integer from 1 to 65536 in NR1 format
```

See Also

- Introduction to :ACQuire Commands
- :ACQuire:TYPE
- :DIGitize
- :WAVeform:COUNt

:ACQuire Commands

:ACQuire:DAALias — N

Command Syntax

```
:ACQuire:DAALias <mode>
```

```
<mode> ::= {DISable | AUTO}
```

The :ACQuire:DAALias command sets the disable anti-alias mode of the oscilloscope.

When set to DISable, anti-alias is always disabled. This is good for cases where dithered data is not desired.

When set to AUTO, the oscilloscope turns off anti-alias control as needed. Such cases are when the FFT or differentiate math functions are silent. The :DIGitize command always turns off the anti-alias control as well.

Query Syntax

```
:ACQuire:DAALias?
```

The :ACQuire:DAALias? query returns the oscilloscope's current disable anti-alias mode setting.

Return Format

```
<mode><NL>
```

```
<mode> ::= {DIS | AUTO}
```

See Also

- Introduction to :ACQuire Commands
- :ACQuire:AALias

:ACQuire Commands

:ACQuire:MODE — C

Command Syntax

```
:ACQuire:MODE <mode>

<mode> ::= {RTIMe | ETIMe}
```

The :ACQuire:MODE command sets the acquisition mode of the oscilloscope. The :ACQuire:MODE RTIMe command sets the oscilloscope in real time mode. This mode is useful to inhibit equivalent time sampling at fast sweep speeds. The :ACQuire:MODE ETIME command sets the oscilloscope in equivalent time mode.

NOTE The obsolete command ACQuire:TYPE:REALtime is functionally equivalent to sending ACQuire:MODE RTIMe; TYPE NORMal.

Query Syntax

```
:ACQuire:MODE?
```

The :ACQuire:MODE? query returns the acquisition mode of the oscilloscope.

Return Format

```
<mode_argument><NL>

<mode_argument> ::= {RTIM | ETIM}
```

See Also

- Introduction to :ACQuire Commands
- :ACQuire:TYPE

:ACQuire Commands

:ACQuire:POINts — C

Query Syntax

```
:ACQuire:POINts?
```

The :ACQuire:POINts? query returns the number of data points that the hardware will acquire from the input signal. The number of points acquired is not directly controllable. To set the number of points to be transferred from the oscilloscope, use the command :WAVeform:POINts. The :WAVeform:POINts? query will return the number of points available to be transferred from the oscilloscope.

Return Format

```
<points_argument><NL>

<points_argument> ::= an integer in NR1 format
```

See Also

- Introduction to :ACQuire Commands
- :DIGitize
- :WAVeform:POINts

:ACQuire Commands

:ACQuire:RSIGnal — N

Command Syntax

```
:ACQuire:RSIGnal <ref_signal_mode>

<ref_signal_mode> ::= {OFF | OUT | IN}
```

The :ACQuire:RSIGnal command selects the 10 MHz reference signal mode.

- The OFF mode disables the oscilloscope's 10 MHz REF BNC connector.
- The OUT mode is used to synchronize the timebase of two or more instruments.
- The IN mode is used to supply a sample clock to the oscilloscope. A 10 MHz square or sine wave signal is input to the BNC connector labeled 10 MHz REF. The amplitude must be between 180 mV and 1 V, with an offset of between 0 V and 2 V.

  CAUTION Do not apply more than ±15 V at the 10 MHz REF BNC connector on the rear panel, or damage to the instrument may occur.

Query Syntax

```
:ACQuire:RSIGnal?
```

The :ACQuire:RSIGnal? query returns the current 10 MHz reference signal mode.

Return Format

```
<ref_signal_mode><NL>

<ref_signal_mode> ::= {OFF | OUT | IN}
```

See Also

- :TIMebase:REFClock
- The *Agilent 6000 Series Oscilloscope User's Guide* for information on using the 10 MHz reference clock.

:ACQuire Commands

:ACQuire:SRATe — N

Query Syntax

```
:ACQuire:SRATe?
```

The :ACQuire:SRATe? query returns the current oscilloscope acquisition sample rate. The sample rate is not directly controllable.

Return Format

```
<sample_rate><NL>

<sample_rate> ::= sample rate in NR3 format
```

See Also

- Introduction to :ACQuire Commands
- :ACQuire:POINts

:ACQuire Commands

:ACQuire:TYPE — C

Command Syntax

```
:ACQuire:TYPE <type>

<type> ::= {NORMal | AVERage | HRESolution | PEAK}
```

The :ACQuire:TYPE command selects the type of data acquisition that is to take place. The acquisition types are: NORMal, AVERage, HRESolution, and PEAK.

- The :ACQuire:TYPE NORMal command sets the oscilloscope in the normal mode.
- The :ACQuire:TYPE AVERage command sets the oscilloscope in the averaging mode. You can set the count by sending the :ACQuire:COUNt command followed by the number of averages. In this mode, the value for averages is an integer from 1 (smoothing) to 65536. The COUNt value determines the number of averages that must be acquired.
- The :ACQuire:TYPE HRESolution command sets the oscilloscope in the high-resolution mode (also known as *smoothing*). This mode is used to reduce noise at slower sweep speeds where the digitizer samples faster than needed to fill memory for the displayed time range.

  For example, if the digitizer samples at 200 MSa/s, but the effective sample rate is 1 MSa/s (because of a slower sweep speed), only 1 out of every 200 samples needs to be stored. Instead of storing one sample (and throwing others away), the 200 samples are averaged together to provide the value for one display point. The slower the sweep speed, the greater the number of samples that are averaged together for each display point.

This command is functionally equivalent to :ACQuire:TYPE AVERage and :ACQuire:COUNt 1.

- The :ACQuire:TYPE PEAK command sets the oscilloscope in the peak detect mode. In this mode, :ACQuire:COUNt has no meaning.

NOTE The obsolete command ACQuire:TYPE:REALtime is functionally equivalent to sending ACQuire:MODE RTIME; TYPE NORMal.

Query Syntax

```
:ACQuire:TYPE?
```

The :ACQuire:TYPE? query returns the current acquisition type.

Return Format

```
<acq_type><NL>

<acq_type> ::= {NORM | AVER | HRES | PEAK}
```

See Also

- Introduction to :ACQuire Commands
- :ACQuire:COUNt
- :ACQuire:MODE
- :DIGitize
- :WAVeform:TYPE
- :WAVeform:PREamble

Example Code

```
' AQUIRE_TYPE - Sets the acquisition mode, which can be NORMAL,
' PEAK, or AVERAGE.
myScope.WriteString ":ACQUIRE:TYPE NORMAL"
```

Example program from the start: VISA COM Example in Visual Basic

Commands by Subsystem

:BUS<n> Commands

Control all oscilloscope functions associated with buses made up of digital channels. See Introduction to :BUS<n> Commands.

Command	Query	Options and Query Returns
:BUS<n>:BIT<m> {{0 | OFF} | {1 | ON}}	:BUS<n>:BIT<m>?	{0 | 1}

		<n> ::= 1 or 2; an integer in NR1 format <m> ::= 0-15; an integer in NR1 format
:BUS<n>:BITS <channel_list>, {{0 | OFF} | {1 | ON}}	:BUS<n>:BITS?	<channel_list>, {0 | 1} <channel_list> ::= (@<m>,<m>:<m> ...) where "," is separator and ":" is range <n> ::= 1 or 2; an integer in NR1 format <m> ::= 0-15; an integer in NR1 format
:BUS<n>:CLEar	n/a	<n> ::= 1 or 2; an integer in NR1 format
:BUS<n>:DISPlay {{0 | OFF} | {1 | ON}}	:BUS<n>:DISPlay?	{0 | 1} <n> ::= 1 or 2; an integer in NR1 format
:BUS<n>:LABel <string>	:BUS<n>:LABel?	<string> ::= quoted ASCII string up to 16 characters <n> ::= 1 or 2; an integer in NR1 format
:BUS<n>:MASK <mask>	:BUS<n>:MASK?	<mask> ::= 32-bit integer in decimal, <nondecimal>, or <string> <nondecimal> ::= #Hnn...n where n ::= {0,..,9 | A,..,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary <string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F} for hexadecimal <n> ::= 1 or 2; an integer in NR1 format

Introduction to :BUS<n> Commands

<n> ::= {1 | 2}

The BUS subsystem commands control the viewing, labeling, and digital channel makeup of two possible buses.

NOTE These commands are only valid for the MSO models.

Reporting the Setup. Use :BUS<n>? to query setup information for the BUS subsystem.

Return Format. The following is a sample response from the :BUS1? query. In this case, the query was issued following a *RST command.

```
:BUS1:DISP 0;LAB "BUS1";MASK +255
```

:BUS<n> Commands

:BUS<n>:BIT<m> — N

Command Syntax

```
:BUS<n>:BIT<m> <display>

<display> ::= {{1 | ON} | {0 | OFF}}

<n> ::= An integer, 1 or 2, is attached as a suffix to BUS
and defines the bus that is affected by the command.

<m> ::= An integer, 0,..,15, is attached as a suffix to BIT
and defines the digital channel that is affected by the command.
```

The :BUS<n>:BIT<m> command includes or excludes the selected bit as part of the definition for the selected bus. If the parameter is a 1 (ON), the bit is included in the definition. If the parameter is a 0 (OFF), the bit is excluded from the definition. *Note:* BIT0-15 correspond to DIGital0-15.

NOTE This command is only valid for the MSO models.

Query Syntax

```
:BUS<n>:BIT<m>?
```

The :BUS<n>:BIT<m>? query returns the value indicating whether the specified bit is included or excluded from the specified bus definition.

Return Format

```
<display><NL>

<display> ::= {0 | 1}
```

See Also

- Introduction to :BUS<n> Commands
- :BUS<n>:BITS
- :BUS<n>:CLEar
- :BUS<n>:DISPlay
- :BUS<n>:LABel
- :BUS<n>:MASK

Example Code

```
' Include digital channel 1 in bus 1:
myScope.WriteString ":BUS1:BIT1 ON"
```

:BUS<n> Commands

:BUS<n>:BITS — N

Command Syntax

```
:BUS<n>:BITS <channel_list>, <display>

<channel_list> ::= (@<m>,<m>:<m>, ...) where commas separate bits and colons define bit ranges

<m> ::= An integer, 0,..,15, defines a digital channel affected by the command.

<display> ::= {{1 | ON} | {0 | OFF}}

<n> ::= An integer, 1 or 2, is attached as a suffix to BUS
and defines the bus that is affected by the command.
```

The :BUS<n>:BITS command includes or excludes the selected bits in the channel list in the definition of the selected bus. If the parameter is a 1 (ON) then the bits in the channel list are included as part of the selected bus definition. If the parameter is a 0 (OFF) then the bits in the channel list are excluded from the definition of the selected bus.

NOTE This command is only valid for the MSO models.

Query Syntax

```
:BUS<n>:BITS?
```

The :BUS<n>:BITS? query returns the definition for the specified bus.

Return Format

```
<channel_list>, <display><NL>

<channel_list> ::= (@<m>,<m>:<m>, ...) where commas separate bits and colons define bit ranges

<display> ::= {0 | 1}
```

See Also

- Introduction to :BUS<n> Commands
- :BUS<n>:BIT<m>
- :BUS<n>:CLEar
- :BUS<n>:DISPlay
- :BUS<n>:LABel
- :BUS<n>:MASK

Example Code

```
' Include digital channels 1, 2, 4, 5, 6, 7, 8, and 9 in bus 1:
myScope.WriteString ":BUS1:BITS (@1,2,4:9), ON"
```

```
' Include digital channels 1, 5, 7, and 9 in bus 1:
myScope.WriteString ":BUS1:BITS (@1,5,7,9), ON"

' Include digital channels 1 through 15 in bus 1:
myScope.WriteString ":BUS1:BITS (@1:15), ON"

' Include digital channels 1 through 5, 8, and 14 in bus 1:
myScope.WriteString ":BUS1:BITS (@1:5,8,14), ON"
```

:BUS<n> Commands

:BUS<n>:CLEar — N

Command Syntax

```
:BUS<n>:CLEar

<n> ::= An integer, 1 or 2, is attached as a suffix to BUS
and defines the bus that is affected by the command.
```

The :BUS<n>:CLEar command excludes all of the digital channels from the selected bus definition.

NOTE This command is only valid for the MSO models.

See Also

- Introduction to :BUS<n> Commands
- :BUS<n>:BIT<m>
- :BUS<n>:BITS
- :BUS<n>:DISPlay
- :BUS<n>:LABel
- :BUS<n>:MASK

:BUS<n> Commands

:BUS<n>:DISPlay — N

Command Syntax

```
:BUS<n>:DISplay <value>

<value> ::= {{1 | ON} | {0 | OFF}}

<n> ::= An integer, 1 or 2, is attached as a suffix to BUS
and defines the bus that is affected by the command.
```

The :BUS<n>:DISPlay command enables or disables the view of the selected bus.

NOTE This command is only valid for the MSO models.

Query Syntax

```
:BUS<n>:DISPlay?
```

The :BUS<n>:DISPlay? query returns the display value of the selected bus.

Return Format

```
<value><NL>

<value> ::= {0 | 1}
```

See Also

- Introduction to :BUS<n> Commands
- :BUS<n>:BIT<m>
- :BUS<n>:BITS
- :BUS<n>:CLEar
- :BUS<n>:LABel
- :BUS<n>:MASK

:BUS<n> Commands

:BUS<n>:LABel — N

Command Syntax

```
:BUS<n>:LABel <quoted_string>

<quoted_string> ::= any series of 16 or less characters as a quoted ASCII string.

<n> ::= An integer, 1 or 2, is attached as a suffix to BUS
and defines the bus that is affected by the command.
```

The :BUS<n>:LABel command sets the bus label to the quoted string. Setting a label for a bus will also result in the name being added to the label list.

NOTE This command is only valid for the MSO models.

NOTE Label strings are 16 characters or less, and may contain any commonly used ASCII characters. Labels with more than 16 characters are truncated to 16 characters.

Query Syntax

```
:BUS<n>:LABel?
```

The :BUS<n>:LABel? query returns the name of the specified bus.

Return Format

```
<quoted_string><NL>
```

```
<quoted_string> ::= any series of 16 or less characters as a quoted ASCII string.
```

See Also

- Introduction to :BUS<n> Commands
- :BUS<n>:BIT<m>
- :BUS<n>:BITS
- :BUS<n>:CLEar
- :BUS<n>:DISPlay
- :BUS<n>:MASK
- :CHANnel<n>:LABel
- :DISPlay:LABList
- :DIGital<n>:LABel

Example Code

```
' Set the bus 1 label to "Data":
myScope.WriteString ":BUS1:LABel 'Data'"
```

:BUS<n> Commands

:BUS<n>:MASK — N

Command Syntax

```
:BUS<n>:MASK <mask>
```

```
<mask> ::= 32-bit integer in decimal, <nondecimal>, or <string>
```

```
<nondecimal> ::= #Hnn...n where n ::= {0,..,9 | A,..,F} for hexadecimal
```

```
<nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary
```

```
<string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F} for hexadecimal
```

```
<n> ::= An integer, 1 or 2, is attached as a suffix to BUS
and defines the bus that is affected by the command.
```

The :BUS<n>:MASK command defines the bits included and excluded in the selected bus according to the mask. Set a mask bit to a "1" to include that bit in the selected bus, and set a mask bit to a "0" to exclude it.

NOTE This command is only valid for the MSO models.

Query Syntax

```
:BUS<n>:MASK?
```

The :BUS<n>:MASK? query returns the mask value for the specified bus.

Return Format

```
<mask><NL> in decimal format
```

See Also

- Introduction to :BUS<n> Commands
- :BUS<n>:BIT<m>
- :BUS<n>:BITS
- :BUS<n>:CLEar
- :BUS<n>:DISPlay
- :BUS<n>:LABel

Commands by Subsystem

:CALibrate Commands

Utility commands for viewing calibration status and for starting the user calibration procedure. See Introduction to :CALibrate Commands.

Command	Query	Options and Query Returns
n/a	:CALibrate:DATE?	<return value> ::= <day>,<month>,<year>; all in NR1 format
:CALibrate:LABel <string>	:CALibrate:LABel?	<string> ::= quoted ASCII string up to 32 characters
:CALibrate:STARt	n/a	n/a
n/a	:CALibrate:STATus?	<return value> ::= ALL,<status_code>,<status_string> <status_code> ::= an integer status code <status_string> ::= an ASCII status string
n/a	:CALibrate:SWITch?	{PROTected | UNPRotected}
n/a	:CALibrate:TEMPerature?	<return value> ::= degrees C delta since last cal in NR3 format
n/a	:CALibrate:TIME?	<return value> ::= <hours>,<minutes>,<seconds>; all in NR1 format

Introduction to :CALibrate Commands

The CALibrate subsystem provides utility commands for:

- Determining the state of the calibration factor protection switch (CAL PROTECT).
- Saving and querying the calibration label string.
- Reporting the calibration time and date.
- Reporting changes in the temperature since the last calibration.
- Starting the user calibration procedure.

:CALibrate Commands

:CALibrate:DATE — N

Query Syntax

```
:CALibrate:DATE?
```

The :CALibrate:DATE? query returns the date of the last calibration.

Return Format

```
<date><NL>

<date> ::= day,month,year in NR1 format<NL>
```

See Also

- Introduction to :CALibrate Commands

:CALibrate Commands

:CALibrate:LABel — N

Command Syntax

```
:CALibrate:LABel <string>

<string> ::= quoted ASCII string of up to 32 characters in length,
not including the quotes
```

The CALibrate:LABel command saves a string that is up to 32 characters in length into the instrument's non-volatile memory. The string may be used to record calibration dates or other information as needed.

Query Syntax

```
:CALibrate:LABel?
```

The :CALibrate:LABel? query returns the contents of the calibration label string.

Return Format

```
<string><NL>
```

```
<string>::= unquoted ASCII string of up to 32 characters in length
```

See Also

- Introduction to :CALibrate Commands

:CALibrate Commands

:CALibrate:STARt — N

Command Syntax

```
:CALibrate:STARt
```

The CALibrate:STARt command starts the user calibration procedure.

NOTE Before starting the user calibration procedure, you must set the rear panel CALIBRATION switch to UNPROTECTED, and you must connect BNC cables from the TRIG OUT connector to the analog channel inputs. See the *User's Guide* for details.

See Also

- Introduction to :CALibrate Commands
- :CALibrate:SWITch

:CALibrate Commands

:CALibrate:STATus — N

Query Syntax

```
:CALibrate:STATus?
```

The :CALibrate:STATus? query returns the summary results of the last user calibration procedure.

Return Format

```
<return value><NL>
```

```
<return value> ::= ALL,<status_code>,<status_string>
```

```
<status_code> ::= an integer status code
```

```
<status_string> ::= an ASCII status string
```

See Also

- Introduction to :CALibrate Commands

:CALibrate Commands

:CALibrate:SWITch — N

Query Syntax

```
:CALibrate:SWITch?
```

The :CALibrate:SWITch? query returns the rear-panel calibration protect (CAL PROTECT) switch state. The value PROTected indicates calibration is disabled, and UNPRotected indicates calibration is enabled.

Return Format

```
<switch><NL>

<switch> ::= {PROT | UNPR}
```

See Also

- Introduction to :CALibrate Commands

:CALibrate Commands

:CALibrate:TEMPerature — N

Query Syntax

```
:CALibrate:TEMPerature?
```

The :CALibrate:TEMPerature? query returns the change in temperature since the last user calibration procedure.

Return Format

```
<return value><NL>

<return value> ::= degrees C delta since last cal in NR3 format
```

See Also

- Introduction to :CALibrate Commands

:CALibrate Commands

:CALibrate:TIME — N

Query Syntax

```
:CALibrate:TIME?
```

The :CALibrate:TIME? query returns the time of the last calibration.

Return Format

```
<date><NL>

<date> ::= hour,minutes,seconds in NR1 format
```

See Also

- Introduction to :CALibrate Commands

Commands by Subsystem

:CHANnel<n> Commands

Control all oscilloscope functions associated with individual analog channels or groups of channels. See Introduction to :CHANnel<n> Commands.

Command	Query	Options and Query Returns
:CHANnel<n>:BWLimit {{0 | OFF} | {1 | ON}}	:CHANnel<n>:BWLimit?	{0 | 1} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:COUPling <coupling>	:CHANnel<n>:COUPling?	<coupling> ::= {AC | DC} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:DISPlay {{0 | OFF} | {1 | ON}}	:CHANnel<n>:DISPlay?	{0 | 1} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:IMPedance <impedance>	:CHANnel<n>:IMPedance?	<impedance> ::= {ONEMeg | FIFTy} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:INVert {{0 | OFF} | {1 | ON}}	:CHANnel<n>:INVert?	{0 | 1} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:LABel <string>	:CHANnel<n>:LABel?	<string> ::= any series of 6 or less ASCII characters enclosed in quotation marks <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:OFFSet <offset> [suffix]	:CHANnel<n>:OFFSet?	<offset> ::= Vertical offset value in NR3 format

		[suffix] ::= {V | mV} <n> ::= 1-2 or 1-4; in NR1 format
:CHANnel<n>:PROBe <attenuation>	:CHANnel<n>:PROBe?	<attenuation> ::= Probe attenuation ratio in NR3 format <n> ::= 1-2 or 1-4r in NR1 format
n/a	:CHANnel<n>:PROBe:ID?	<probe id> ::= unquoted ASCII string up to 11 characters <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:PROBe:SKEW <skew_value>	:CHANnel<n>:PROBe:SKEW?	<skew_value> ::= -100 ns to +100 ns in NR3 format <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:PROBe:STYPe <signal type>	:CHANnel<n>:PROBe:STYPe?	<signal type> ::= {DIFFerential | SINGle} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:PROTection	:CHANnel<n>:PROTection?	{NORM | TRIP} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:RANGe <range> [suffix]	:CHANnel<n>:RANGe?	<range> ::= Vertical full-scale range value in NR3 format [suffix] ::= {V | mV} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:SCALe <scale> [suffix]	:CHANnel<n>:SCALe?	<scale> ::= Vertical units per division value in NR3 format [suffix] ::= {V | mV} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:UNITs <units>	:CHANnel<n>:UNITs?	<units> ::= {VOLTs | AMPeres} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:VERNier {{0 | OFF} | {1 | ON}}	:CHANnel<n>:VERNier?	{0 | 1} <n> ::= 1-2 or 1-4 in NR1 format

Introduction to :CHANnel<n> Commands

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models

The CHANnel<n> subsystem commands control an analog channel (vertical or Y-axis of the oscilloscope). Channels are independently programmable for all offset, probe, coupling, bandwidth limit, inversion, vernier, and range (scale) functions. The channel number (1, 2, 3, or 4) specified in the command selects the analog channel that is affected by the command.

A label command provides identifying annotations of up to 6 characters.

You can toggle the channel displays on and off with the :CHANnel<n>:DISPlay command as well as with the root level commands :VIEW and :BLANk.

NOTE The obsolete CHANnel subsystem is supported.

Reporting the Setup. Use :CHANnel1?, :CHANnel2?, :CHANnel3? or :CHANnel4? to query setup information for the CHANnel<n> subsystem.

Return Format. The following are sample responses from the :CHANnel<n>? query. In this case, the query was issued following a *RST command.

```
:CHAN1:RANG +40.0E+00;OFFS +0.00000E+00;COUP DC;IMP ONEM;DISP 1;BWL 0;
INV 0;LAB "1";UNIT VOLT;PROB +10E+00;PROB:SKEW +0.00E+00;STYP SING
```

:CHANnel<n> Commands

:CHANnel<n>:BWLimit — C

Command Syntax

```
:CHANnel<n>:BWLimit <bwlimit>
```

```
<bwlimit> ::= {{1 | ON} | {0 | OFF}
```

```
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
```

```
<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :CHANnel<n>:BWLimit command controls an internal low-pass filter. When the filter is on, the bandwidth of the specified channel is limited to approximately 25 MHz.

Query Syntax

```
:CHANnel<n>:BWLimit?
```

The :CHANnel<n>:BWLimit? query returns the current setting of the low-pass filter.

Return Format

```
<bwlimit><NL>
```

```
<bwlimit> ::= {1 | 0}
```

See Also

- Introduction to :CHANnel<n> Commands

:CHANnel<n> Commands

:CHANnel<n>:COUPling — C

Command Syntax

```
:CHANnel<n>:COUPling <coupling>

<coupling> ::= {AC | DC}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :CHANnel<n>:COUPling command selects the input coupling for the specified channel. The coupling for each analog channel can be set to AC or DC.

Query Syntax

```
:CHANnel<n>:COUPling?
```

The :CHANnel<n>:COUPling? query returns the current coupling for the specified channel.

Return Format

```
<coupling value><NL>

<coupling value> ::= {AC | DC}
```

See Also

- Introduction to :CHANnel<n> Commands

:CHANnel<n> Commands

:CHANnel<n>:DISPlay — C

Command Syntax

```
:CHANnel<n>:DISPlay <display value>

<display value> ::= {{1 | ON} | {0 | OFF}}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :CHANnel<n>:DISPlay command turns the display of the specified channel on or off.

Query Syntax

```
:CHANnel<n>:DISPlay?
```

The :CHANnel<n>:DISPlay? query returns the current display setting for the specified channel.

Return Format

```
<display value><NL>

<display value> ::= {1 | 0}
```

See Also

- Introduction to :CHANnel<n> Commands
- :VIEW
- :BLANk
- :STATus
- :POD<n>:DISPlay
- :DIGital<n>:DISPlay

:CHANnel<n> Commands

:CHANnel<n>:IMPedance — C

Command Syntax

```
:CHANnel<n>:IMPedance <impedance>

<impedance> ::= {ONEMeg | FIFTy}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :CHANnel<n>:IMPedance command selects the input impedance setting for the specified analog channel. The legal values for this command are ONEMeg (1 MΩ) and FIFTy (50Ω).

NOTE The analog channel input impedance of the 100 MHz bandwidth oscilloscope models is fixed at ONEMeg (1 MΩ).

Query Syntax

```
:CHANnel<n>:IMPedance?
```

The :CHANnel<n>:IMPedance? query returns the current input impedance setting for the specified channel.

Return Format

```
<impedance value><NL>
```

```
<impedance value> ::= {ONEM | FIFT}
```

See Also

- Introduction to :CHANnel<n> Commands

:CHANnel<n> Commands

:CHANnel<n>:INVert — N

Command Syntax

```
:CHANnel<n>:INVert <invert value>

<invert value> ::= {{1 | ON} | {0 | OFF}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :CHANnel<n>:INVert command selects whether or not to invert the input signal for the specified channel. The inversion may be 1 (ON/inverted) or 0 (OFF/not inverted).

Query Syntax

```
:CHANnel<n>:INVert?
```

The :CHANnel<n>:INVert? query returns the current state of the channel inversion.

Return Format

```
<invert value><NL>

<invert value> ::= {0 | 1}
```

See Also

- Introduction to :CHANnel<n> Commands

:CHANnel<n> Commands

:CHANnel<n>:LABel — N

Command Syntax

```
:CHANnel<n>:LABel <string>

<string> ::= quoted ASCII string

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
```

```
<n> ::= {1 | 2} for the two channel oscilloscope models
```

NOTE Label strings are six characters or less, and may contain any commonly used ASCII characters. Labels with more than 6 characters are truncated to six characters. Lower case characters are converted to upper case.

The :CHANnel<n>:LABel command sets the analog channel label to the string that follows. Setting a label for a channel also adds the name to the label list in non-volatile memory (replacing the oldest label in the list).

Query Syntax

```
:CHANnel<n>:LABel?
```

The :CHANnel<n>:LABel? query returns the label associated with a particular analog channel.

Return Format

```
<string><NL>

<string> ::= quoted ASCII string
```

See Also

- Introduction to :CHANnel<n> Commands
- :DISPlay:LABel
- :DIGital<n>:LABel
- :DISPlay:LABList
- :BUS<n>:LABel

Example Code

```
   ' LABEL - This command allows you to write a name (six characters
   ' maximum) next to the channel number.  It is not necessary, but
   ' can be useful for organizing the display.
  myScope.WriteString ":CHANNEL1:LABEL ""CAL 1"""    ' Label channel1 "CAL 1".
  myScope.WriteString ":CHANNEL2:LABEL ""CAL2"""    ' Label channel1 "CAL2".
```

Example program from the start: VISA COM Example in Visual Basic

:CHANnel<n> Commands

:CHANnel<n>:OFFSet — C

Command Syntax

```
:CHANnel<n>:OFFSet <offset> [<suffix>]

<offset> ::= Vertical offset value in NR3 format

<suffix> ::= {V | mV}
```

```
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
```

```
<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :CHANnel<n>:OFFSet command sets the value that is represented at center screen for the selected channel. The range of legal values varies with the value set by the :CHANnel<n>:RANGe and :CHANnel<n>:SCALe commands. If you set the offset to a value outside of the legal range, the offset value is automatically set to the nearest legal value. Legal values are effected by the probe attenuation setting.

Query Syntax

```
:CHANnel<n>:OFFSet?
```

The :CHANnel<n>:OFFSet? query returns the current offset value for the selected channel.

Return Format

```
<offset><NL>
```

```
<offset> ::= Vertical offset value in NR3 format
```

See Also

- Introduction to :CHANnel<n> Commands
- :CHANnel<n>:RANGe
- :CHANnel<n>:SCALe
- :CHANnel<n>:PROBe

:CHANnel<n> Commands

:CHANnel<n>:PROBe — C

Command Syntax

```
:CHANnel<n>:PROBe <attenuation>
```

```
<attenuation> ::= probe attenuation ratio in NR3 format
```

```
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
```

```
<n> ::= {1 | 2} for the two channel oscilloscope models
```

```
The obsolete attenuation values X1, X10, X20, X100 are also supported.
```

The :CHANnel<n>:PROBe command specifies the probe attenuation factor for the selected channel. The probe attenuation factor may be 0.1 to 1000. This command does not change the actual input sensitivity of the oscilloscope. It changes the reference constants for scaling the display factors, for making automatic measurements, and for setting trigger levels.

If an AutoProbe probe is connected to the oscilloscope, the attenuation value cannot be changed from the sensed value. Attempting to set the oscilloscope to an attenuation value other than the sensed value produces an error.

Query Syntax

```
:CHANnel<n>:PROBe?
```

The :CHANnel<n>:PROBe? query returns the current probe attenuation factor for the selected channel.

Return Format

```
<attenuation><NL>

<attenuation> ::= probe attenuation ratio in NR3 format
```

See Also

- Introduction to :CHANnel<n> Commands
- :CHANnel<n>:RANGe
- :CHANnel<n>:SCALe
- :CHANnel<n>:OFFSet

Example Code

```
   ' CHANNEL_PROBE - Sets the probe attenuation factor for the selected
   ' channel.  The probe attenuation factor may be set from 0.1 to 1000.
   myScope.WriteString ":CHAN1:PROBE 10"   ' Set Probe to 10:1.
```

Example program from the start: VISA COM Example in Visual Basic

:CHANnel<n> Commands

:CHANnel<n>:PROBe:ID — C

Query Syntax

```
:CHANnel<n>:PROBe:ID?

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :CHANnel<n>:PROBe:ID? query returns the type of probe attached to the specified oscilloscope channel.

Return Format

```
<probe id><NL>

<probe id> ::= unquoted ASCII string up to 11 characters
```

Some of the possible returned values are:

- 1131A
- 1132A
- 1134A
- 1147A
- 1153A
- 1154A
- 1156A
- 1157A
- 1158A
- 1159A
- AutoProbe
- E2621A
- E2622A
- E2695A
- E2697A
- HP1152A
- HP1153A
- NONE
- Probe
- Unknown
- Unsupported

See Also

- Introduction to :CHANnel<n> Commands

:CHANnel<n> Commands

:CHANnel<n>:PROBe:SKEW — C

Command Syntax

```
:CHANnel<n>:PROBe:SKEW <skew value>

<skew value> ::= skew time in NR3 format

<skew value> ::= -100 ns to +100 ns

<n> ::= {1 | 2 | 3 | 4}
```

The :CHANnel<n>:PROBe:SKEW command sets the channel-to-channel skew factor for the specified channel. Each analog channel can be adjusted + or -100 ns for a total of 200 ns difference between channels. You can use the oscilloscope's probe skew control to remove cable-delay errors between channels.

Query Syntax

```
:CHANnel<n>:PROBe:SKEW?
```

The :CHANnel<n>:PROBe:SKEW? query returns the current probe skew setting for the selected channel.

Return Format

```
<skew value><NL>

<skew value> ::= skew value in NR3 format
```

See Also

- Introduction to :CHANnel<n> Commands

:CHANnel<n> Commands

:CHANnel<n>:PROBe:STYPe — C

Command Syntax

NOTE This command is valid only for the 113xA Series probes.

```
:CHANnel<n>:PROBe:STYPe <signal type>

<signal type> ::= {DIFFerential | SINGle}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :CHANnel<n>:PROBe:STYPe command sets the channel probe signal type (STYPe) to differential or single-ended when using the 113xA Series probes and determines how offset is applied.

When single-ended is selected, the :CHANnel<n>:OFFset command changes the offset value of the probe amplifier. When differential is selected, the :CHANnel<n>:OFFset command changes the offset value of the channel amplifier.

Query Syntax

```
:CHANnel<n>:PROBe:STYPe?
```

The :CHANnel<n>:PROBe:STYPe? query returns the current probe signal type setting for the selected channel.

Return Format

```
<signal type><NL>

<signal type> ::= {DIFF | SING}
```

See Also

- Introduction to :CHANnel<n> Commands
- :CHANnel<n>:OFFSet

:CHANnel<n> Commands

:CHANnel<n>:PROTection — N

Command Syntax

```
:CHANnel<n>:PROTection[:CLEar]
```

```
<n> ::= {1 | 2 | 3 | 4}
```

When the analog channel input impedance is set to 50Ω (on the 300 MHz, 500 MHz, and 1 GHz bandwidth oscilloscope models), the input channels are protected against overvoltage. When an overvoltage condition is sensed, the input impedance for the channel is automatically changed to 1 MΩ. The :CHANnel<n>:PROTection [:CLEar] command is used to clear (reset) the overload protection. It allows the channel to be used again in 50Ω mode after the signal that caused the overload has been removed from the channel input. Reset the analog channel input impedance to 50Ω (see :CHANnel<n>:IMPedance) after clearing the overvoltage protection.

Query Syntax

```
:CHANnel<n>:PROTection?
```

The :CHANnel<n>:PROTection query returns the state of the input protection for CHANnel<n>. If a channel input has experienced an overload, TRIP (tripped) will be returned; otherwise NORM (normal) is returned.

Return Format

```
{NORM | TRIP}<NL>
```

See Also

- Introduction to :CHANnel<n> Commands
- :CHANnel<n>:COUPling
- :CHANnel<n>:IMPedance
- :CHANnel<n>:PROBe

:CHANnel<n> Commands

:CHANnel<n>:RANGe — C

Command Syntax

```
:CHANnel<n>:RANGe <range>[<suffix>]
```

```
<range> ::= vertical full-scale range value in NR3 format

<suffix> ::= {V | mV}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :CHANnel<n>:RANGe command defines the full-scale vertical axis of the selected channel. When using 1:1 probe attenuation, the range can be set to any value from:

- 8 mV to 40 V for the 100 MHz models.
- 16 mV to 8 V for the 300 MHz – 1 GHz models with the input impedance set to 50Ω.

If the probe attenuation is changed, the range value is multiplied by the probe attenuation factor.

Query Syntax

```
:CHANnel<n>:RANGe?
```

The :CHANnel<n>:RANGe? query returns the current full-scale range setting for the specified channel.

Return Format

```
<range_argument><NL>

<range_argument> ::= vertical full-scale range value in NR3 format
```

See Also

- Introduction to :CHANnel<n> Commands
- :CHANnel<n>:SCALe
- :CHANnel<n>:PROBe

Example Code

```
   ' CHANNEL_RANGE - Sets the full scale vertical range in volts.  The
   ' range value is 8 times the volts per division.
   myScope.WriteString ":CHANNEL1:RANGE 8"   ' Set the vertical range to 8 volts.
```

Example program from the start: VISA COM Example in Visual Basic

:CHANnel<n> Commands

:CHANnel<n>:SCALe — N

Command Syntax

```
:CHANnel<n>:SCALe <scale>[<suffix>]

<scale> ::= vertical units per division in NR3 format
```

```
<suffix> ::= {V | mV}
```

```
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
```

```
<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :CHANnel<n>:SCALe command sets the vertical scale, or units per division, of the selected channel. When using 1:1 probe attenuation, legal values for the scale range from:

- 1 mV to 5 V for the 100 MHz models.
- 2 mV to 1 V for the 300 MHz – 1 GHz models with the input impedance set to 50Ω.

If the probe attenuation is changed, the scale value is multiplied by the probe's attenuation factor.

Query Syntax

```
:CHANnel<n>:SCALe?
```

The :CHANnel<n>:SCALe? query returns the current scale setting for the specified channel.

Return Format

```
<scale value><NL>
```

```
<scale value> ::= vertical units per division in NR3 format
```

See Also

- Introduction to :CHANnel<n> Commands
- :CHANnel<n>:RANGe
- :CHANnel<n>:PROBe

:CHANnel<n> Commands

:CHANnel<n>:UNITs — N

Command Syntax

```
:CHANnel<n>:UNITs <units>
```

```
<units> ::= {VOLTs | AMPeres}
```

```
<n> ::= {1 | 2} for the two channel oscilloscope models
```

```
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
```

The :CHANnel<n>:UNITs command sets the measurement units for the connected probe. Select VOLTs for a voltage probe and select AMPeres for a current probe. Measurement results, channel sensitivity, and trigger level will reflect the measurement units you select.

Query Syntax

```
:CHANnel<n>:UNITs?
```

The :CHANnel<n>:UNITs? query returns the current units setting for the specified channel.

Return Format

```
<units><NL>

<units> ::= {VOLT | AMP}
```

See Also

- Introduction to :CHANnel<n> Commands
- :CHANnel<n>:RANGe
- :CHANnel<n>:PROBe
- :EXTernal:UNITs

:CHANnel<n> Commands

:CHANnel<n>:VERNier — N

Command Syntax

```
:CHANnel<n>:VERNier <vernier value>

<vernier value> ::= {{1 | ON} | {0 | OFF}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :CHANnel<n>:VERNier command specifies whether the channel's vernier (fine vertical adjustment) setting is ON (1) or OFF (0).

Query Syntax

```
:CHANnel<n>:VERNier?
```

The :CHANnel<n>:VERNier? query returns the current state of the channel's vernier setting.

Return Format

```
<vernier value><NL>

<vernier value> ::= {0 | 1}
```

See Also

- Introduction to :CHANnel<n> Commands

Commands by Subsystem

Common (*) Commands

Commands defined by IEEE 488.2 standard that are common to all instruments. See Introduction to Common (*) Commands.

<table>
<tr><th>Command</th><th>Query</th><th>Options and Query Returns</th></tr>
<tr><td>*CLS</td><td>n/a</td><td>n/a</td></tr>
<tr><td>*ESE &lt;mask&gt;</td><td>*ESE?</td><td>&lt;mask&gt; ::= 0 to 255; an integer in NR1 format:
<table>
<tr><th>Bit</th><th>Weight</th><th>Name</th><th>Enables</th></tr>
<tr><td>7</td><td>128</td><td>PON</td><td>Power On</td></tr>
<tr><td>6</td><td>64</td><td>URQ</td><td>User Request</td></tr>
<tr><td>5</td><td>32</td><td>CME</td><td>Command Error</td></tr>
<tr><td>4</td><td>16</td><td>EXE</td><td>Execution Error</td></tr>
<tr><td>3</td><td>8</td><td>DDE</td><td>Device Dependent Error</td></tr>
<tr><td>2</td><td>4</td><td>QYE</td><td>Query Error</td></tr>
<tr><td>1</td><td>2</td><td>RQL</td><td>Request Control</td></tr>
<tr><td>0</td><td>1</td><td>OPC</td><td>Operation Complete</td></tr>
</table></td></tr>
<tr><td>n/a</td><td>*ESR?</td><td>&lt;status> ::= 0 to 255; an integer in NR1 format</td></tr>
<tr><td>n/a</td><td>*IDN?</td><td>AGILENT TECHNOLOGIES,<model>, <serial number>,X.XX.XX
<model> ::= the model number of the instrument
<serial number> ::= the serial number of the instrument
<X.XX.XX> ::= the software revision of the instrument</td></tr>
<tr><td>n/a</td><td>*LRN?</td><td>&lt;learn_string> ::= current instrument setup as a block of data in IEEE 488.2 # format</td></tr>
<tr><td>*OPC</td><td>*OPC?</td><td>ASCII "1" is placed in the output queue when all pending device operations have completed.</td></tr>
<tr><td>n/a</td><td>*OPT?</td><td>&lt;return_value> ::= 0,0,<license info>
<license info> ::= <All field>, <reserved>, <Factory MSO>, <Upgraded MSO>, <Probe field>, <Memory>, <Low Speed Serial>, <reserved>, <reserved>
<All field> ::= {0 | All}
<reserved> ::= 0
<Factory MSO> ::= {0 | MSO}
<Upgraded MSO> ::= {0 | MSO}</td></tr>
</table>

<table>
<tr><td></td><td></td><td>&lt;Probe field> ::= 0
<Memory> ::= {0 | mem2M | mem8M}
<Low Speed Serial> ::= {0 | LSS}
<reserved> ::= 0
<reserved&gt; ::= 0</td></tr>
<tr><td>*RCL &lt;value&gt;</td><td>n/a</td><td>&lt;value> ::= {0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9}</td></tr>
<tr><td>*RST</td><td>n/a</td><td>See *RST (Reset)</td></tr>
<tr><td>*SAV &lt;value&gt;</td><td>n/a</td><td>&lt;value> ::= {0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9}</td></tr>
<tr><td>*SRE &lt;mask&gt;</td><td>*SRE?</td><td>&lt;mask&gt; ::= sum of all bits that are set, 0 to 255; an integer in NR1 format. <mask> ::= following values:
<table>
<tr><th>Bit</th><th>Weight</th><th>Name</th><th>Enables</th></tr>
<tr><td>7</td><td>128</td><td>OPER</td><td>Operation Status Register</td></tr>
<tr><td>6</td><td>64</td><td>----</td><td>(Not used.)</td></tr>
<tr><td>5</td><td>32</td><td>ESB</td><td>Event Status Bit</td></tr>
<tr><td>4</td><td>16</td><td>MAV</td><td>Message Available</td></tr>
<tr><td>3</td><td>8</td><td>----</td><td>(Not used.)</td></tr>
<tr><td>2</td><td>4</td><td>MSG</td><td>Message</td></tr>
<tr><td>1</td><td>2</td><td>USR</td><td>User</td></tr>
<tr><td>0</td><td>1</td><td>TRG</td><td>Trigger</td></tr>
</table></td></tr>
<tr><td>n/a</td><td>*STB?</td><td>&lt;value> ::= 0 to 255; an integer in NR1 format, as shown in the following:
<table>
<tr><th>Bit</th><th>Weight</th><th>Name</th><th>"1" Indicates</th></tr>
<tr><td>7</td><td>128</td><td>OPER</td><td>An enabled operation status condition occurred.</td></tr>
<tr><td>6</td><td>64</td><td>RQS/ MSS</td><td>Instrument is requesting service.</td></tr>
<tr><td>5</td><td>32</td><td>ESB</td><td>Enabled event status condition occurred.</td></tr>
<tr><td>4</td><td>16</td><td>MAV</td><td>An output message is ready.</td></tr>
<tr><td>3</td><td>8</td><td>----</td><td>(Not used.)</td></tr>
<tr><td>2</td><td>4</td><td>MSG</td><td>Message has been displayed.</td></tr>
<tr><td>1</td><td>2</td><td>USR</td><td>An enabled user event condition has occurred.</td></tr>
<tr><td>0</td><td>1</td><td>TRG</td><td>A trigger occurred.</td></tr>
</table></td></tr>
<tr><td>*TRG</td><td>n/a</td><td>n/a</td></tr>
<tr><td>n/a</td><td>*TST?</td><td>&lt;result> ::= 0 or non-zero value; an integer in NR1 format</td></tr>
</table>

*WAI	n/a	n/a

Introduction to Common (*) Commands

The common commands are defined by the IEEE 488.2 standard. They are implemented by all instruments that comply with the IEEE 488.2 standard. They provide some of the basic instrument functions, such as instrument identification and reset, reading the instrument setup, and determining how status is read and cleared.

Common commands can be received and processed by the instrument whether they are sent over the interface as separate program messages or within other program messages. If an instrument subsystem has been selected and a common command is received by the instrument, the instrument remains in the selected subsystem. For example, if the program message ":ACQuire:TYPE AVERage; *CLS; COUNt 256" is received by the instrument, the instrument sets the acquire type, then clears the status information and sets the average count.

In contrast, if a root level command or some other subsystem command is within the program message, you must re-enter the original subsystem after the command. For example, the program message ":ACQuire:TYPE AVERage; :AUToscale; :ACQuire:COUNt 256" sets the acquire type, completes the autoscale, then sets the acquire count. In this example, :ACQuire must be sent again after the :AUToscale command in order to re-enter the ACQuire subsystem and set the count.

NOTE Each of the status registers has an enable (mask) register. By setting the bits in the enable register, you can select the status information you want to use.

Common (*) Commands

*CLS (Clear Status) — C

Command Syntax

```
*CLS
```

The *CLS common command clears the status data structures, the device-defined error queue, and the Request-for-OPC flag.

NOTE If the *CLS command immediately follows a program message terminator, the output queue and the MAV (message available) bit are cleared.

See Also

- Introduction to Common (*) Commands
- *STB (Read Status Byte)
- *ESE (Standard Event Status Enable)
- *ESR (Standard Event Status Register)
- *SRE (Service Request Enable)
- :SYSTem:ERRor

Common (*) Commands

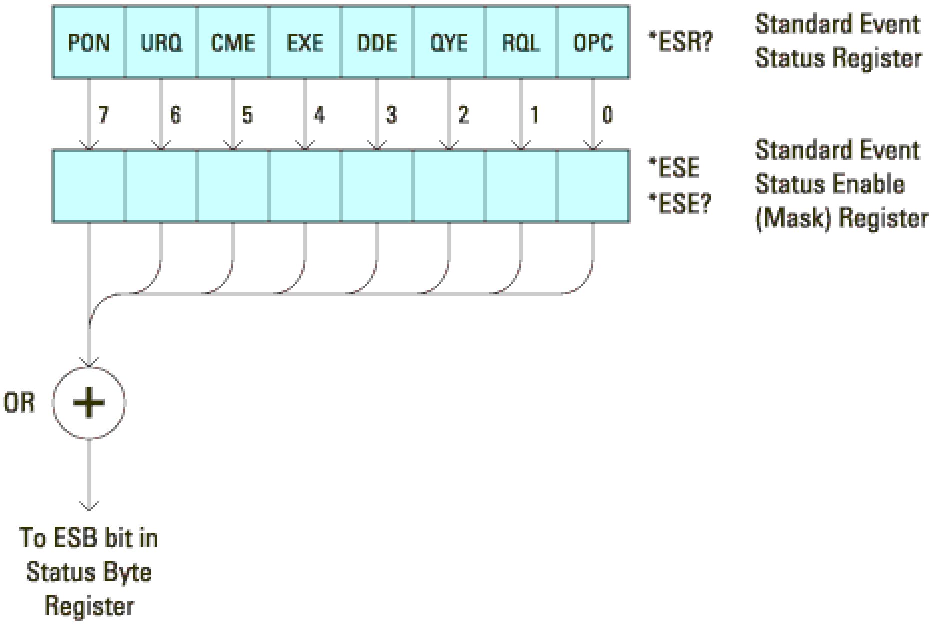
*ESE (Standard Event Status Enable) — C

Command Syntax

```
*ESE <mask_argument>

<mask_argument> ::= integer from 0 to 255
```

The *ESE common command sets the bits in the Standard Event Status Enable Register. The Standard Event Status Enable Register contains a mask value for the bits to be enabled in the Standard Event Status Register. A "1" in the Standard Event Status Enable Register enables the corresponding bit in the Standard Event Status Register. A zero disables the bit.


PON
URQ
CME
EXE
DDE
QYE
RQL
OPC
*ESR?
Standard Event Status Register
7
6
5
4
3
2
1
0
*ESE
*ESE?
Standard Event Status Enable (Mask) Register
OR
To ESB bit in Status Byte Register

Standard Event Status Enable (ESE):

Bit	Name	Description	When Set (1 = High = True), Enables:
7	PON	Power On	Event when an OFF to ON transition occurs.
6	URQ	User Request	Event when a front-panel key is pressed.
5	CME	Command Error	Event when a command error is detected.
4	EXE	Execution Error	Event when an execution error is detected.
3	DDE	Device Dependent Error	Event when a device-dependent error is detected.
2	QYE	Query Error	Event when a query error is detected.
1	RQL	Request Control	Event when the device is requesting control. (Not used.)
0	OPC	Operation Complete	Event when an operation is complete.

Query Syntax

```
*ESE?
```

The *ESE? query returns the current contents of the Standard Event Status Enable Register.

Return Format

```
<mask_argument><NL>

<mask_argument> ::= 0,..,255; an integer in NR1 format.
```

See Also

- Introduction to Common (*) Commands
- *ESR (Standard Event Status Register)
- *OPC (Operation Complete)
- *CLS (Clear Status)

Common (*) Commands

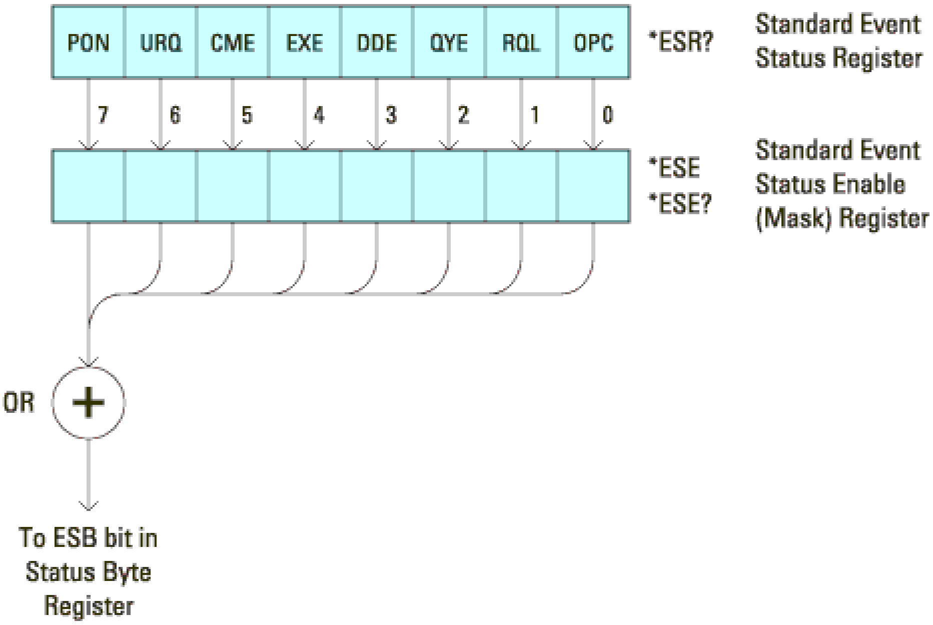
*ESR (Standard Event Status Register) — C

Query Syntax

```
*ESR
```

The *ESR? query returns the contents of the Standard Event Status Register. When you read the Event Status Register, the value returned is the total bit weights of all of the bits that are high at the time you read the byte. Reading the register clears the Event Status Register.

The following table shows bit weight, name, and condition for each bit.


PON
URQ
CME
EXE
DDE
QYE
RQL
OPC
*ESR?
Standard Event Status Register
7
6
5
4
3
2
1
0
*ESE
*ESE?
Standard Event Status Enable (Mask) Register
OR
To ESB bit in Status Byte Register

Standard Event Status Register (ESR):

Bit	Name	Description	When Set (1 = High = True), Indicates:
7	PON	Power On	An OFF to ON transition has occurred.
6	URQ	User Request	A front-panel key has been pressed.
5	CME	Command Error	A command error has been detected.
4	EXE	Execution Error	An execution error has been detected.
3	DDE	Device Dependent Error	A device-dependent error has been detected.
2	QYE	Query Error	A query error has been detected.
1	RQL	Request Control	The device is requesting control. (Not used.)
0	OPC	Operation Complete	Operation is complete.

Return Format

```
<status><NL>

<status> ::= 0,..,255; an integer in NR1 format.
```

NOTE Reading the Standard Event Status Register clears it. High or 1 indicates the bit is true.

See Also

- Introduction to Common (*) Commands
- *ESE (Standard Event Status Enable)
- *OPC (Operation Complete)

- *CLS (Clear Status)
- :SYSTem:ERRor

Common (*) Commands

*IDN (Identification Number) — C

Query Syntax

```
*IDN?
```

The *IDN? query identifies the instrument type and software version.

Return Format

```
AGILENT TECHNOLOGIES,<model>,<serial number>,X.XX.XX <NL>

<model> ::= the model number of the instrument

<serial number> ::= the serial number of the instrument

X.XX.XX ::= the software revision of the instrument
```

See Also

- Introduction to Common (*) Commands
- *OPT (Option Identification)

Common (*) Commands

*LRN (Learn Device Setup) — C

Query Syntax

```
*LRN?
```

The *LRN? query result contains the current state of the instrument. This query is similar to the :SYSTem:SETup? query, except that it contains ":SYST:SET " before the binary block data. The query result is a valid command that can be used to restore instrument settings at a later time.

Return Format

```
<learn_string><NL>

<learn_string> ::= :SYST:SET <setup_data>

<setup_data> ::= binary block data in IEEE 488.2 # format
```

<learn string> specifies the current instrument setup. The block size is subject to change with different firmware revisions.

NOTE The *LRN? query return format for the 6000 Series oscilloscopes has changed from previous Agilent oscilloscopes to match the IEEE 488.2 specification which says that the query result must contain ":SYST:SET " before the binary block data.

See Also

- Introduction to Common (*) Commands
- *RCL (Recall)
- *SAV (Save)
- :SYSTem:SETup

Common (*) Commands

*OPC (Operation Complete) — C

Command Syntax

```
*OPC
```

The *OPC command sets the operation complete bit in the Standard Event Status Register when all pending device operations have finished.

Query Syntax

```
*OPC?
```

The *OPC? query places an ASCII "1" in the output queue when all pending device operations have completed. The interface hangs until this query returns.

Return Format

```
<complete><NL>

<complete> ::= 1
```

See Also

- Introduction to Common (*) Commands
- *ESE (Standard Event Status Enable)
- *ESR (Standard Event Status Register)
- *CLS (Clear Status)

Common (*) Commands

*OPT (Option Identification) — C

Query Syntax

```
*OPT?
```

The *OPT? query reports the options installed in the instrument. This query returns a string that identifies the module and its software revision level.

Return Format

```
0,0,<license info>

<license info> ::= <All field>,<reserved>,<Factory MSO>,<Upgraded MSO>,<Probe field>,
                   <Memory>,<Low Speed Serial>,<reserved>,<reserved>

<All field> ::= {0 | All}

<reserved> ::= 0

<Factory MSO> ::= {0 | MSO}

<Upgraded MSO> ::= {0 | MSO}

<Probe field> ::= 0

<Memory> ::= {0 | mem2M | mem8M}

<Low Speed Serial> ::= {0 | LSS}

<reserved> ::= 0

<reserved> ::= 0

The <Factory MSO> <Upgraded MSO> fields indicate whether the unit is a mixed-signal
oscilloscope and, if so, whether it was factory installed or upgraded from an analog
channels only oscilloscope (DSO).
```

The *OPT? query returns the following:

Module	Module Id
No modules attached	0,0,0,0,MSO,0,0,mem8M,0,0,0

See Also

- Introduction to Common (*) Commands
- *IDN (Identification Number)

Common (*) Commands

*RCL (Recall) — C

Command Syntax

```
*RCL <value>
```

```
<value> ::= {0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9}
```

The *RCL command restores the state of the instrument from the specified save/recall register.

See Also

- Introduction to Common (*) Commands
- *SAV (Save)

Common (*) Commands

*RST (Reset) — C

Command Syntax

```
*RST
```

The *RST command places the instrument in a known state. Reset conditions are:

Acquire Menu

Mode	Normal
Realtime	On
Averaging	Off
# Averages	8

Analog Channel Menu

Channel 1	On
Channel 2	Off
Volts/division	5.00 V
Offset	0.00
Coupling	DC
Probe attenuation	AutoProbe (if AutoProbe is connected), otherwise 1.0:1
Vernier	Off
Invert	Off
BW limit	Off
Impedance	1 M Ohm
Units	Volts
Skew	0

Cursor Menu

Source	Channel 1

Digital Channel Menu (MSO models only)	
Channel 0 - 15	Off
Labels	Off
Threshold	TTL (1.4V)

Display Menu	
Definite persistence	Off
Grid	33%
Vectors	On

Quick Meas Menu	
Source	Channel 1

Run Control
Scope is running

Time Base Menu	
Main time/division	100 us
Main time base delay	0.00 s
Delay time/division	500 ns
Delay time base delay	0.00 s
Reference	center
Mode	main
Vernier	Off

Trigger Menu	
Type	Edge
Mode	Auto
Coupling	dc
Source	Channel 1
Level	0.0 V
Slope	Positive
HF Reject and noise reject	Off

Holdoff	60 ns
External probe attenuation	AutoProbe (if AutoProbe is connected), otherwise 1.0:1
External Units	Volts
External Impedance	1 M Ohm

See Also

- Introduction to Common (*) Commands

Example Code

```
   ' RESET - This command puts the oscilloscope into a known state.
   ' This statement is very important for programs to work as expected.
   ' Most of the following initialization commands are initialized by
   ' *RST.  It is not necessary to reinitialize them unless the default
   ' setting is not suitable for your application.
  myScope.WriteString "*RST"   ' Reset the oscilloscope to the defaults.
```

Example program from the start: VISA COM Example in Visual Basic

Common (*) Commands

*SAV (Save) — C

Command Syntax

```
*SAV<value>

<value> ::= {0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9}
```

The *SAV command stores the current state of the instrument in a save register. The data parameter specifies the register where the data will be saved.

See Also

- Introduction to Common (*) Commands
- *RCL (Recall)

Common (*) Commands

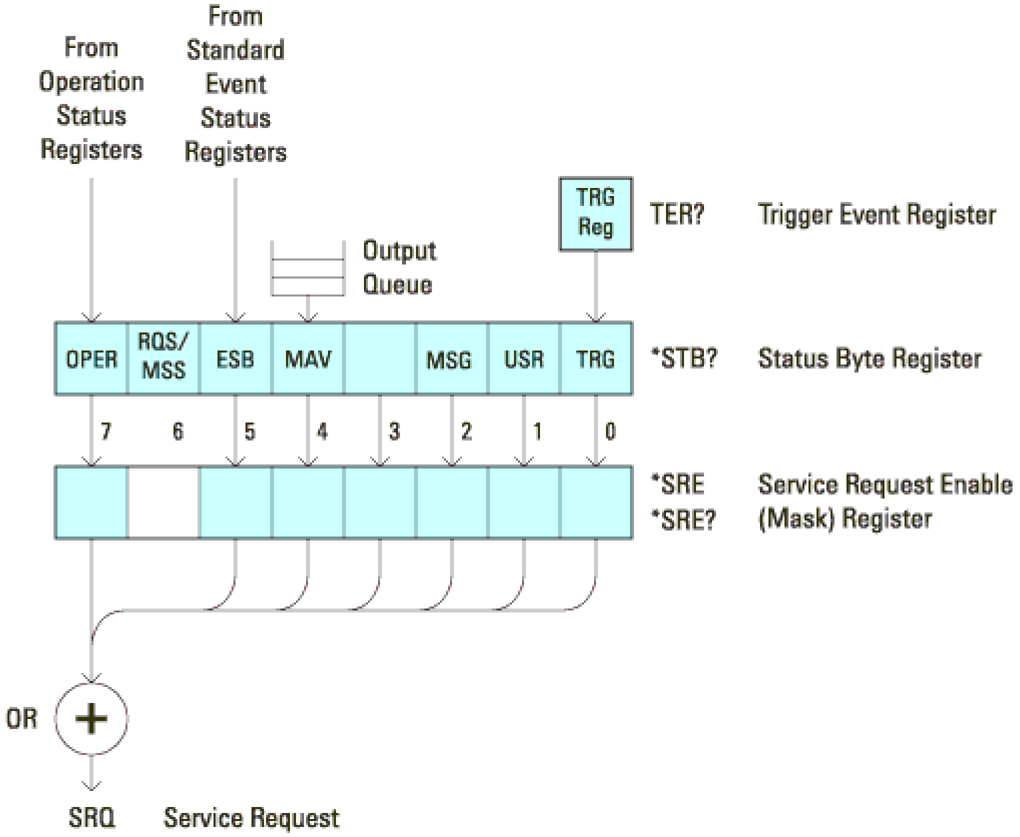
*SRE (Service Request Enable) — C

Command Syntax

```
*SRE <mask>

<mask> ::= integer with values defined in the following table.
```

The *SRE command sets the bits in the Service Request Enable Register. 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 enables the corresponding bit in the Status Byte Register. A zero disables the bit.



Service Request Enable Register (SRE):

Bit	Name	Description	When Set (1 = High = True), Enables:
7	OPER	Operation Status Register	Interrupts when enabled conditions in the Operation Status Register (OPER) occur.
6	---	---	(Not used.)
5	ESB	Event Status Bit	Interrupts when enabled conditions in the Standard Event Status Register (ESR) occur.
4	MAV	Message Available	Interrupts when messages are in the Output Queue.
3	---	---	(Not used.)
2	MSG	Message	Interrupts when an advisory has been displayed on the oscilloscope.
1	USR	User Event	Interrupts when enabled user event conditions occur.
0	TRG	Trigger	Interrupts when a trigger occurs.

Query Syntax

*SRE?

The *SRE? query returns the current value of the Service Request Enable Register.

Return Format

```
<mask><NL>
```

```
<mask> ::= sum of all bits that are set, 0,..,255; an integer in NR1 format
```

See Also

- Introduction to Common (*) Commands
- *STB (Read Status Byte)
- *CLS (Clear Status)

Common (*) Commands

*STB (Read Status Byte) — C

Query Syntax

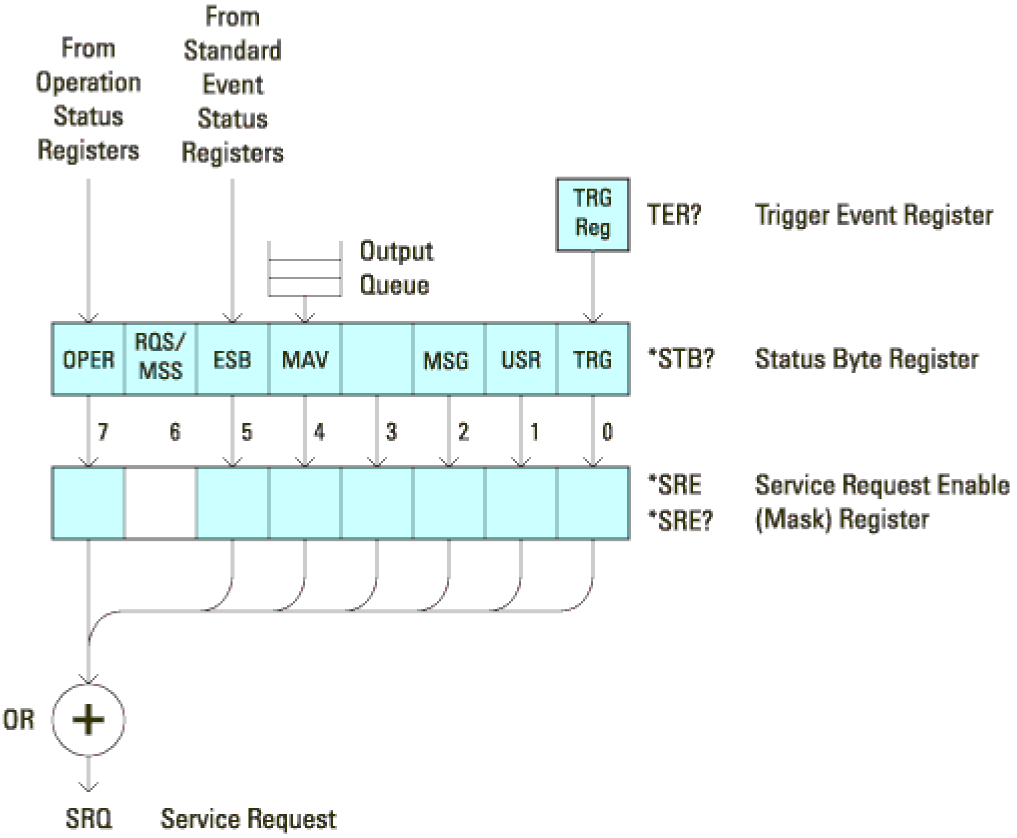
```
*STB?
```

The *STB? query returns the current value of the instrument's status byte. The MSS (Master Summary Status) bit is reported on bit 6 instead of the RQS (request service) bit. The MSS indicates whether or not the device has at least one reason for requesting service.

Return Format

```
<value><NL>
```

```
<value> ::= 0,..,255; an integer in NR1 format
```



Status Byte Register (STB):

Bit	Name	Description	When Set (1 = High = True), Indicates:
7	OPER	Operation Status Register	An enabled condition in the Operation Status Register (OPER) has occurred.
6	RQS	Request Service	When polled, that the device is requesting service.
	MSS	Master Summary Status	When read (by *STB?), whether the device has a reason for requesting service.
5	ESB	Event Status Bit	An enabled condition in the Standard Event Status Register (ESR) has occurred.
4	MAV	Message Available	There are messages in the Output Queue.
3	---	---	(Not used, always 0.)
2	MSG	Message	An advisory has been displayed on the oscilloscope.
1	USR	User Event	An enabled user event condition has occurred.
0	TRG	Trigger	A trigger has occurred.

NOTE To read the instrument's status byte with RQS reported on bit 6, use the interface Serial Poll.

See Also

- Introduction to Common (*) Commands
- *SRE (Service Request Enable)

Common (*) Commands

*TRG (Trigger) — C

Command Syntax

```
*TRG
```

The *TRG command has the same effect as the :DIGitize command with no parameters.

See Also

- Introduction to Common (*) Commands
- :DIGitize
- :RUN
- :STOP

Common (*) Commands

*TST (Self Test) — C

Query Syntax

```
*TST?
```

The *TST? query performs a self-test on the instrument. The result of the test is placed in the output queue. A zero indicates the test passed and a non-zero indicates the test failed. If the test fails, refer to the troubleshooting section of the *Service Guide*.

Return Format

```
<result><NL>

<result> ::= 0 or non-zero value; an integer in NR1 format
```

See Also

- Introduction to Common (*) Commands

Common (*) Commands

*WAI (Wait To Continue) — C

Command Syntax

```
*WAI
```

The *WAI command has no function in the oscilloscope, but is parsed for compatibility with other instruments.

See Also

- Introduction to Common (*) Commands

Commands by Subsystem

:DIGital<n> Commands

Control all oscilloscope functions associated with individual digital channels. See Introduction to :DIGital<n> Commands.

Command	Query	Options and Query Returns
:DIGital<n>:DISPlay {{0 | OFF} | {1 | ON}}	:DIGital<n>:DISPlay?	{0 | 1} <n> ::= 0-15; an integer in NR1 format
:DIGital<n>:LABel <string>	:DIGital<n>:LABel?	<string> ::= any series of 6 or less ASCII characters enclosed in quotation marks <n> ::= 0-15; an integer in NR1 format
:DIGital<n>:POSition <position>	:DIGital<n>:POSition?	<n> ::= 0-15; an integer in NR1 format <position> ::= 0-7 if display size = large, 0-15 if size = medium, 0-31 if size = small
:DIGital<n>:SIZE <value>	:DIGital<n>:SIZE?	<value> ::= {SMALl | MEDium | LARGe}

:DIGital<n>:THReshold <value>[suffix]	:DIGital<n>:THReshold?	<n> ::= 0-15; an integer in NR1 format
		<value> ::= {CMOS | ECL | TTL | <user defined value>}
		<user defined value> ::= value in NR3 format from -8.00 to +8.00
		[suffix] ::= {V | mV | uV}

Introduction to :DIGital<n> Commands

```
<n> ::= {0,..,15}
```

The DIGital subsystem commands control the viewing, labeling, and positioning of digital channels. They also control threshold settings for groups of digital channels (D0-D7, D8-D15).

These commands are only valid for the MSO models.

NOTE

Reporting the Setup. Use :DIGital<n>? to query setup information for the DIGital subsystem.

Return Format. The following is a sample response from the :DIGital0? query. In this case, the query was issued following a *RST command.

```
:DIG0:DISP 0;THR +1.40E+00;LAB 'D0';POS +0
```

:DIGital<n> Commands

:DIGital<n>:DISPlay — N

Command Syntax

```
:DIGital<n>:DISPlay <display>

<display> ::= {{1 | ON} | {0 | OFF}}

<n> ::= An integer, 0, 1,..,15, is attached as a suffix to the command
and defines the logic channel that is affected by the command.
```

The :DIGital<n>:DISPlay command turns digital display on or off for the specified channel.

NOTE This command is only valid for the MSO models.

Query Syntax

```
:DIGital<n>:DISPlay?
```

The :DIGital<n>:DISPlay? query returns the current digital display setting for the specified channel.

Return Format

```
<display><NL>

<display> ::= {0 | 1}
```

See Also

- Introduction to :DIGital<n> Commands
- :POD<n>:DISPlay
- :CHANnel<n>:DISPlay
- :VIEW
- :BLANk
- :STATus

:DIGital<n>:LABel — N

Command Syntax

```
:DIGital<n>:LABel <string>
```

```
<string> ::= any series of 6 or less characters as a quoted ASCII string.
```

```
<n> ::= An integer, 0,..,15, is attached as a suffix to the command and
defines the logic channel that is affected by the command.
```

The :DIGital<n>:LABel command sets the channel label to the string that follows. Setting a label for a channel also adds the name to the label list in non-volatile memory (replacing the oldest label in the list).

NOTE This command is only valid for the MSO models.

NOTE Label strings are six characters or less, and may contain any commonly used ASCII characters. Labels with more than 6 characters are truncated to six characters.

Query Syntax

```
:DIGital<n>:LABel?
```

The :DIGital<n>:LABel? query returns the name of the specified channel.

Return Format

```
<label string><NL>
```

```
<label string> ::= any series of 6 or less characters as a quoted ASCII string.
```

See Also

- Introduction to :DIGital<n> Commands
- :CHANnel<n>:LABel
- :DISPlay:LABList
- :BUS<n>:LABel

:DIGital<n> Commands

:DIGital<n>:POSition — N

Command Syntax

```
:DIGital<n>:POSition <position>
```

<position> ::= integer in NR1 format.

<n> ::= An integer, 0, 1,..,15, is attached as a suffix to the command and
defines the logic channel that is affected by the command.

Channel Size	Position	Top	Bottom
Large	0-7	7	0
Medium	0-15	15	0
Small	0-31	31	0

The :DIGital<n>:POSition command sets the position of the specified channel.

NOTE This command is only valid for the MSO models.

Query Syntax

```
:DIGital<n>:POSition?
```

The :DIGital<n>:POSition? query returns the position of the specified channel.

Return Format

```
<position><NL>
```

<position> ::= integer in NR1 format.

See Also

- Introduction to :DIGital<n> Commands

:DIGital<n> Commands

:DIGital<n>:SIZE — N

Command Syntax

```
:DIGital<n>:SIZE <value>
```

<n> ::= An integer, 0, 1,..,15, is attached as a suffix to the command
and defines the logic channel that is affected by the command.

<value> ::= {SMALl | MEDium | LARGe}

The :DIGital<n>:SIZE command specifies the size of digital channels on the display. Sizes are set for all digital channels. Therefore, if you set the size on digital channel 0 (for example), the same size is set on channels 1 through 15 as well.

This command is only valid for the MSO models.

NOTE

Query Syntax

```
:DIGital<n>:SIZE?
```

The :DIGital<n>:SIZE? query returns the size setting for the specified digital channels.

Return Format

```
<size_value><NL>

<size_value> ::= {SMAL | MED | LARG}
```

See Also

- Introduction to :DIGital<n> Commands
- :POD<n>:SIZE
- :DIGital<n>:POSition

:DIGital<n> Commands

:DIGital<n>:THReshold — N

Command Syntax

```
:DIGital<n>:THReshold <value>

<value> ::= {CMOS | ECL | TTL | <user defined value>[<suffix>]}

<user defined value> ::= -8.00 to +8.00 in NR3 format

<suffix> ::= {V | mV | uV}

<n> ::= An integer, 0, 1,..,15, is attached as a suffix to the command
and defines the logic channel that is affected by the command.
```

- TTL = 1.4V
- CMOS = 2.5V
- ECL = -1.3V

The :DIGital<n>:THReshold command sets the logic threshold value for all channels grouped with the specified channel (D0-D7, D8-D15). The threshold is used for triggering purposes and for displaying the digital data as high (above the threshold) or low (below the threshold).

NOTE This command is only valid for the MSO models.

Query Syntax

```
:DIGital<n>:THReshold?
```

The :DIGital<n>:THReshold? query returns the threshold value for the specified channel.

Return Format

```
<value><NL>

<value> ::= threshold value in NR3 format
```

See Also

- Introduction to :DIGital<n> Commands
- :POD<n>:THReshold
- :TRIGger[:EDGE]:LEVel

Commands by Subsystem

:DISPlay Commands

Control how waveforms, graticule, and text are displayed and written on the screen. See Introduction to :DISPlay Commands.

Command	Query	Options and Query Returns
:DISPlay:CLEar	n/a	n/a
:DISPlay:DATA [<format>][,][<area>][,][<palette>]<display data>	:DISPlay:DATA? [<format>][,][<area>][,][<palette>]	<format> ::= {TIFF} (command) <area> ::= {GRATicule} (command) <palette> ::= {MONochrome} (command) <format> ::= {TIFF | BMP | BMP8bit | PNG} (query) <area> ::= {GRATicule | SCReen} (query) <palette> ::= {MONochrome | GRAYscale | COLor} (query) <display data> ::= data in IEEE 488.2 # format
:DISPlay:LABel {{0 | OFF} | {1 | ON}}	:DISPlay:LABel?	{0 | 1}
:DISPlay:LABList <binary block>	:DISPlay:LABList?	<binary block> ::= an ordered list of up to 75 labels, each 6 characters maximum, separated by

		newline characters
:DISPlay:PERSistence <value>	:DISPlay:PERSistence?	<value> ::= {MINimum | INFinite}}
:DISPlay:SOURce <value>	:DISPlay:SOURce?	<value> ::= {PMEMory{0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9}}
:DISPlay:VECTors {{1 | ON} | {0 | OFF}}	:DISPlay:VECTors?	{1 | 0}

Introduction to :DISPlay Commands

The DISPlay subsystem is used to control the display storage and retrieval of waveform data, labels, and text. This subsystem allows the following actions:

- Clear the waveform area on the display.
- Turn vectors on or off.
- Set waveform persistence.
- Specify labels.
- Save and Recall display data.

Reporting the Setup. Use :DISPlay? to query the setup information for the DISPlay subsystem.

Return Format. The following is a sample response from the :DISPlay? query. In this case, the query was issued following a *RST command.

```
:DISP:LAB 0;CONN 1;PERS MIN;SOUR PMEM9
```

:DISPlay Commands

:DISPlay:CLEar — N

Command Syntax

```
:DISPlay:CLEar
```

The :DISPlay:CLEar command clears the display and resets all associated measurements. If the oscilloscope is stopped, all currently displayed data is erased. If the oscilloscope is running, all of the data for active channels and functions is erased; however, new data is displayed on the next acquisition.

See Also

- Introduction to :DISPlay Commands
- :CDISplay

:DISPlay Commands

:DISPlay:DATA — N

Command Syntax

```
:DISPlay:DATA [<format>][,][<area>][,][<palette>]<display data>

<format> ::= {TIFF}

<area> ::= {GRATicule}

<palette> ::= {MONochrome}

<display data> ::= binary block data in IEEE-488.2 # format.
```

The :DISPlay:DATA command writes trace memory data (a display bitmap) to the display or to one of the trace memories in the instrument.

If a data format or area is specified, the :DISPlay:DATA command transfers the data directly to the display. If neither the data format nor the area is specified, the command transfers data to the trace memory specified by the :DISPlay:SOURce command. Available trace memories are PMEMory0-9 and these memories correspond to the INTERN_0-9 files in the front panel Save/Recall menu.

Graticule data is a low resolution bitmap of the graticule area in TIFF format. This is the same data saved using the front panel Save/Recall menu or the *SAV (Save) command.

Query Syntax

```
:DISPlay:DATA? [<format>][,] [<area>][,] [<palette>]

<format> ::= {TIFF | BMP | BMP8bit | PNG}

<area> ::= {GRATicule | SCReen}

<palette> ::= {MONochrome | GRAYscale | COLor}
```

The :DISPlay:DATA? query reads display data from the screen or from one of the trace memories in the instrument. The format for the data transmission is the # format defined in the IEEE 488.2 specification.

If a data format or area is specified, the :DISPlay:DATA query transfers the data directly from the display. If neither the data format nor the area is specified, the query transfers data from the trace memory specified by the :DISPlay:SOURce command.

Screen data is the full display and is high resolution in grayscale or color. It may be read from the instrument in 24-bit bmp, 8-bit bmp, or 24-bit png format. This data cannot be sent back to the instrument.

Graticule data is a low resolution bitmap of the graticule area in TIFF format. You can get this data and send it back to the oscilloscope.

NOTE If the format is TIFF, the only valid value area parameter is GRATicule, and the only valid palette parameter is MONOchrome.

If the format is something other than TIFF, the only valid area parameter is SCReen, and the only valid values for palette are GRAYscale or COLor.

Return Format

```
<display data><NL>

<display data> ::= binary block data in IEEE-488.2 # format.
```

See Also

- Introduction to :DISPlay Commands
- :DISPlay:SOURce
- :MERGe
- :PRINt
- *RCL (Recall)
- *SAV (Save)
- :VIEW

Example Code

```
' IMAGE_TRANSFER - In this example, we will query for the image data
' with ":DISPLAY:DATA?", read the data, and then save it to a file.
Dim byteData() As Byte
myScope.IO.Timeout = 15000
myScope.WriteString ":DISPLAY:DATA? BMP, SCREEN, COLOR"
byteData = myScope.ReadIEEEBlock(BinaryType_UI1)
' Output display data to a file:
strPath = "c:\scope\data\screen.bmp"
' Remove file if it exists.
If Len(Dir(strPath)) Then
  Kill strPath
End If
Close #1   ' If #1 is open, close it.
Open strPath For Binary Access Write Lock Write As #1   ' Open file for output.
Put #1, , byteData   ' Write data.
Close #1   ' Close file.
myScope.IO.Timeout = 5000
```

Example program from the start: VISA COM Example in Visual Basic

:DISPlay Commands

:DISPlay:LABel — N

Command Syntax

```
:DISPlay:LABel <value>

<value> ::= {{1 | ON} | {0 | OFF}}
```

The :DISPlay:LABel command turns the analog and digital channel labels on and off.

Query Syntax

```
:DISPlay:LABel?
```

The :DISPlay:LABel? query returns the display mode of the analog and digital labels.

Return Format

```
<value><NL>

<value> ::= {0 | 1}
```

See Also

- Introduction to :DISPlay Commands
- :CHANnel<n>:LABel

Example Code

```
   ' DISP_LABEL (not executed in this example)
   '  - Turns label names ON or OFF on the analyzer display.
   myScope.WriteString ":DISPLAY:LABEL ON"   ' Turn on labels.
```

Example program from the start: VISA COM Example in Visual Basic

:DISPlay Commands

:DISPlay:LABList — N

Command Syntax

```
:DISPlay:LABList <binary block data>

<binary block> ::= an ordered list of up to 75 labels, a maximum of six
characters each, separated by newline characters.
```

The :DISPlay:LABList command sets the label list. The labels are added in alphabetical order.

Query Syntax

```
:DISPlay:LABList?
```

The :DISPlay:LABList? query returns the alphabetical label list.

Return Format

```
<binary block><NL>

<binary block> ::= an ordered list of up to 75 labels, a maximum of six
characters each, separated by newline characters.
```

See Also

- Introduction to :DISPlay Commands
- :DISPlay:LABel
- :CHANnel<n>:LABel
- :DIGital<n>:LABel

:DISPlay Commands

:DISPlay:PERSistence — N

Command Syntax

```
:DISPlay:PERSistence <value>
```

```
<value> ::= {MINimum | INFinite}
```

The :DISPlay:PERSistence command specifies the persistence setting. MINimum indicates zero persistence and INFinite indicates infinite persistence. Use the :DISPlay:CLEar or :CDISplay root command to erase points stored by infinite persistence.

Query Syntax

```
:DISPlay:PERSistence?
```

The :DISPlay:PERSistence? query returns the specified persistence value.

Return Format

```
<value><NL>
```

```
<value> ::= {MIN | INF}
```

See Also

- Introduction to :DISPlay Commands
- :DISPlay:CLEar
- :CDISplay

:DISPlay Commands

:DISPlay:SOURce — N

Command Syntax

```
:DISPlay:SOURce <value>
```

```
<value> ::= {PMEMory0 | PMEMory1 | PMEMory2 | PMEMory3 | PMEMory4
| PMEMory5 | PMEMory6 | PMEMory7 | PMEMory8 | PMEMory9}
```

```
PMEMory0-9 ::= pixel memory 0 through 9
```

The :DISPlay:SOURce command specifies the default source and destination for the :DISPlay:DATA command and query. PMEMory0-9 correspond to the INTERN_0-9 files found in the front panel Save/Recall menu.

Query Syntax

```
:DISPlay:SOURce?
```

The :DISPlay:SOURce? query returns the specified SOURce.

Return Format

```
<value><NL>

<value> ::= {PMEM0 | PMEM1 | PMEM2 | PMEM3 | PMEM4 | PMEM5 | PMEM6
| PMEM7 | PMEM8 | PMEM9}
```

See Also

- Introduction to :DISPlay Commands
- :DISPlay:DATA

:DISPlay Commands

:DISPlay:VECTors — N

Command Syntax

```
:DISPlay:VECTors <vectors>

<vectors> ::= {{1 | ON} | {0 | OFF}}
```

The :DISPlay:VECTors command turns vector display on or off. When vectors are turned on, the oscilloscope displays lines connecting sampled data points. When vectors are turned off, only the sampled data is displayed.

Query Syntax

```
:DISPlay:VECTors?
```

The :DISPlay:VECTors? query returns whether vector display is on or off.

Return Format

```
<vectors><NL>

<vectors> ::= {1 | 0}
```

See Also

- Introduction to :DISPlay Commands

Commands by Subsystem

:EXTernal Trigger Commands

Control the input characteristics of the external trigger input. See Introduction to :EXTernal Trigger Commands.

Command	Query	Options and Query Returns
:EXTernal:BWLimit <bwlimit>	:EXTernal:BWLimit?	<bwlimit> ::= {0 | OFF}
:EXTernal:IMPedance <value>	:EXTernal:IMPedance?	<impedance> ::= {ONEMeg | FIFTy}
:EXTernal:PROBe <attenuation>	:EXTernal:PROBe?	<attenuation> ::= probe attenuation ratio in NR3 format
n/a	:EXTernal:PROBe:ID?	<probe id> ::= unquoted ASCII string up to 11 characters
:EXTernal:PROBe:STYPe <signal type>	:EXTernal:PROBe:STYPe?	<signal type> ::= {DIFFerential | SINGle}
:EXTernal:PROTection[:CLEar]	:EXTernal:PROTection?	{NORM | TRIP}
:EXTernal:RANGe <range> [<suffix>]	:EXTernal:RANGe?	<range> ::= vertical full-scale range value in NR3 format <suffix> ::= {V | mV}
:EXTernal:UNITs <units>	:EXTernal:UNITs?	<units> ::= {VOLTs | AMPeres}

Introduction to :EXTernal Trigger Commands

The EXTernal trigger subsystem commands control the input characteristics of the external trigger input. The probe factor, impedance, input range, input protection state, units, and bandwidth limit settings may all be queried. Depending on the instrument type, some settings may be changeable.

Reporting the Setup. Use :EXTernal? to query setup information for the EXTernal subsystem.

Return Format. The following is a sample response from the :EXTernal query. In this case, the query was issued following a *RST command.

```
:EXT:BWL 0;IMP ONEM;RANG +8.0E+00;UNIT VOLT;PROB +1.0E+00;PROB:STYP SING
```

:EXTernal Trigger Commands

:EXTernal:BWLimit — C

Command Syntax

```
:EXTernal:BWLimit <bwlimit>

<bwlimit> ::= {0 | OFF}
```

The :EXTernal:BWLimit command is provided for product compatibility. The only legal value is 0 or OFF. Use the :TRIGger:HFReject command to limit bandwidth on the external trigger input.

Query Syntax

```
:EXTernal:BWLimit?
```

The :EXTernal:BWLimit? query returns the current setting of the low-pass filter (always 0).

Return Format

```
<bwlimit><NL>

<bwlimit> ::= 0
```

See Also

- Introduction to :EXTernal Trigger Commands
- Introduction to :TRIGger Commands
- :TRIGger:HFReject

:EXTernal Trigger Commands

:EXTernal:IMPedance — C

Command Syntax

```
:EXTernal:IMPedance <value>

<value> ::= {ONEMeg | FIFTy}
```

The :EXTernal:IMPedance command selects the input impedance setting for the external trigger. The legal values for this command are ONEMeg (1 MΩ) and FIFTy (50Ω).

NOTE You can set external trigger input impedance to FIFTy (50Ω) on the 2-channel, 300 MHz, 500 MHz, and 1 GHz bandwidth oscilloscope models.

Query Syntax

```
:EXTernal:IMPedance?
```

The :EXTernal:IMPedance? query returns the current input impedance setting for the external trigger.

Return Format

```
<impedance value><NL>

<impedance value> ::= {ONEM | FIFT}
```

See Also

- Introduction to :EXTernal Trigger Commands
- Introduction to :TRIGger Commands
- :CHANnel<n>:IMPedance

:EXTernal Trigger Commands

:EXTernal:PROBe — C

Command Syntax

```
:EXTernal:PROBe <attenuation>
```

```
<attenuation> ::= probe attenuation ratio in NR3 format
```

The :EXTernal:PROBe command specifies the probe attenuation factor for the external trigger. The probe attenuation factor may be 0.1 to 1000. This command does not change the actual input sensitivity of the oscilloscope. It changes the reference constants for scaling the display factors and for setting trigger levels.

If an AutoProbe probe is connected to the oscilloscope, the attenuation value cannot be changed from the sensed value. Attempting to set the oscilloscope to an attenuation value other than the sensed value produces an error.

Query Syntax

```
:EXTernal:PROBe?
```

The :EXTernal:PROBe? query returns the current probe attenuation factor for the external trigger.

Return Format

```
<attenuation><NL>
```

```
<attenuation> ::= probe attenuation ratio in NR3 format
```

See Also

- Introduction to :EXTernal Trigger Commands
- :EXTernal:RANGe
- Introduction to :TRIGger Commands
- :CHANnel<n>:PROBe

:EXTernal Trigger Commands

:EXTernal:PROBe:ID — C

Query Syntax

```
:EXTernal:PROBe:ID?
```

The :EXTernal:PROBe:ID? query returns the type of probe attached to the external trigger input.

Return Format

```
<probe id><NL>
```

```
<probe id> ::= unquoted ASCII string up to 11 characters
```

Some of the possible returned values are:

- 1131A
- 1132A
- 1134A
- 1147A
- 1153A
- 1154A
- 1156A
- 1157A
- 1158A
- 1159A
- AutoProbe
- E2621A
- E2622A
- E2695A
- E2697A
- HP1152A
- HP1153A
- NONE
- Probe
- Unknown
- Unsupported

See Also

- Introduction to :EXTernal Trigger Commands

:EXTernal Trigger Commands

:EXTernal:PROBe:STYPe — C

Command Syntax

NOTE This command is valid only for the 113xA Series probes.

```
:EXTernal:PROBe:STYPe <signal type>

<signal type> ::= {DIFFerential | SINGle}
```

The :EXTernal:PROBe:STYPe command sets the external trigger probe signal type (STYPe) to differential or single-ended when using the 113xA Series probes and determines how offset is applied.

Query Syntax

```
:EXTernal:PROBe:STYPe?
```

The :EXTernal:PROBe:STYPe? query returns the current probe signal type setting for the external trigger.

Return Format

```
<signal type><NL>

<signal type> ::= {DIFF | SING}
```

See Also

- Introduction to :EXTernal Trigger Commands

:EXTernal Trigger Commands

:EXTernal:PROTection — N

Command Syntax

```
:EXTernal:PROTection[:CLEar]
```

When the external trigger input impedance is set to 50Ω (on the 2-channel, 300 MHz, 500 MHz, and 1 GHz bandwidth oscilloscope models), the external trigger input is protected against overvoltage. When an overvoltage condition is sensed, the input impedance for the external trigger is automatically changed to 1 MΩ. The :EXTernal:PROTection[:CLEar] command is used to clear (reset) the overload protection. It allows the external trigger to be used again in 50Ω mode after the signal that caused the overload has been removed from the external trigger input. Reset the external trigger input impedance to 50Ω (see :EXTernal:IMPedance) after clearing the overvoltage protection.

Query Syntax

```
:EXTernal:PROTection?
```

The :EXTernal:PROTection query returns the state of the input protection for external trigger. If the external trigger input has experienced an overload, TRIP (tripped) will be returned; otherwise NORM (normal) is returned.

Return Format

```
{NORM | TRIP}<NL>
```

See Also

- Introduction to :EXTernal Trigger Commands
- :EXTernal:IMPedance
- :EXTernal:PROBe

:EXTernal Trigger Commands

:EXTernal:RANGe — C

Command Syntax

```
:EXTernal:RANGe <range>[<suffix>]

<range> ::= vertical full-scale range value in NR3 format

<suffix> ::= {V | mV}
```

The :EXTernal:RANGe command is provided for product compatibility. The range can only be set to 5.0 V when using 1:1 probe attenuation. If the probe attenuation is changed, the range value is multiplied by the probe attenuation factor.

Query Syntax

```
:EXTernal:RANGe?
```

The :EXTernal:RANGe? query returns the current full-scale range setting for the external trigger.

Return Format

```
<range_argument><NL>

<range_argument> ::= external trigger range value in NR3 format = (5.0 V) * (probe attenuatio
```

See Also

- Introduction to :EXTernal Trigger Commands
- :EXTernal:PROBe
- Introduction to :TRIGger Commands

:EXTernal Trigger Commands

:EXTernal:UNITs — N

Command Syntax

```
:EXTernal:UNITs <units>

<units> ::= {VOLTs | AMPeres}
```

The :EXTernal:UNITs command sets the measurement units for the probe connected to the external trigger input. Select VOLTs for a voltage probe and select AMPeres for a current probe. Measurement results, channel sensitivity, and trigger level will reflect the measurement units you select.

Query Syntax

```
:EXTernal:UNITs?
```

The :CHANnel<n>:UNITs? query returns the current units setting for the external trigger.

Return Format

```
<units><NL>

<units> ::= {VOLT | AMP}
```

See Also

- Introduction to :EXTernal Trigger Commands
- Introduction to :TRIGger Commands
- :EXTernal:RANGe
- :EXTernal:PROBe
- :CHANnel<n>:UNITs

Commands by Subsystem

:FUNCtion Commands

Control functions in the measurement/storage module. See Introduction to :FUNCtion Commands.

Command	Query	Options and Query Returns
:FUNCtion:CENTer <frequency>	:FUNCtion:CENTer?	<frequency> ::= the current center frequency in NR3 format. The range of legal values is from 0 Hz to 25 GHz.
:FUNCtion:DISPlay {{0 | OFF} | {1 | ON}}	:FUNCtion:DISPlay?	{0 | 1}
:FUNCtion:OFFSet <offset>	:FUNCtion:OFFSet?	<offset> ::= the value at center screen in NR3 format. The range of legal values is +/-10 times the current sensitivity of the selected function.
:FUNCtion:OPERation <operation>	:FUNCtion:OPERation?	<operation> ::= {SUBTract | MULTiply | INTegrate | DIFFerentiate | FFT}
:FUNCtion:RANGe <range>	:FUNCtion:RANGe?	<range> ::= the full-scale vertical axis value in NR3 format. The range for ADD, SUBT, MULT is 8E-6 to 800E+3. The range for the INTegrate function is 8E-9 to 400E+3. The range for the DIFFerentiate function is 80E-3 to 8.0E12 (depends on current sweep speed). The range for the FFT function is 8 to 800 dBV.
:FUNCtion:REFerence	:FUNCtion:REFerence?	<level> ::= the current reference level in

<level>		NR3 format. The range of legal values is from 400.0 dBV to +400.0 dBV (depending on current range value).
:FUNCtion:SCALe <scale value>[<suffix>]	:FUNCtion:SCALe?	<scale value> ::= integer in NR1 format <suffix> ::= {V | dB}
:FUNCtion:SOURce <source>	:FUNCtion:SOURce?	<source> ::= {CHANnel<n> | ADD | SUBT | MULT} <n> ::= 1-2 or 1-4 in NR1 format
:FUNCtion:SPAN <span>	:FUNCtion:SPAN?	<span> ::= the current frequency span in NR3 format. Legal values are 1 Hz to 100 GHz.
:FUNCtion:WINDow <window>	:FUNCtion:WINDow?	<window> ::= {RECTangular | HANNing | FLATtop}

Introduction to :FUNCtion Commands

The FUNCtion subsystem controls the math functions in the oscilloscope. Multiply (channel 1 x channel 2), subtract (channel 1 - channel 2), differentiate, integrate, and FFT (Fast Fourier Transform) operations are available. These math operations only use the analog (vertical) channels.

NOTE To perform analog channel addition, set analog channel 2 to invert and select subtract (channel 1 - channel 2).

The SOURce, DISPlay, RANGe, and OFFSet commands apply to any function. The SPAN, CENTer, and WINDow commands are only useful for FFT functions. When FFT is selected, the cursors change from volts and time to decibels (dB) and frequency (Hz).

Reporting the Setup. Use :FUNCtion? to query setup information for the FUNCtion subsystem.

Return Format. The following is a sample response from the :FUNCtion? queries. In this case, the query was issued following a *RST command.

```
:FUNC:OPER SUBT;DISP 0;RANG +8.00E+00;OFFS +0.00000E+00
```

:FUNCtion Commands

:FUNCtion:CENTer —

Command Syntax

```
:FUNCtion:CENTer <frequency>
```

```
<frequency> ::= the current center frequency in NR3 format.  The range
of legal values is from 0 Hz to 25 GHz.
```

The :FUNCtion:CENTer command sets the center frequency when FFT (Fast Fourier Transform) is selected.

Query Syntax

```
:FUNCtion:CENTer?
```

The :FUNCtion:CENTer? query returns the current center frequency in Hertz.

Return Format

```
<frequency><NL>

<frequency> ::= the current center frequency in NR3 format.  The range
of legal values is from 0 Hz to 25 GHz.
```

NOTE After a *RST (Reset) or :AUToscale command, the values returned by the :FUNCtion:CENTer? and :FUNCtion:SPAN? queries depend on the current :TIMebase:RANGe value. Once you change either the :FUNCtion:CENTer or :FUNCtion:SPAN value, they no longer track the :TIMebase:RANGe value.

See Also

- Introduction to :FUNCtion Commands
- :FUNCtion:SPAN
- :TIMebase:RANGe
- :TIMebase:SCALe

:FUNCtion Commands

:FUNCtion:DISPlay — N

Command Syntax

```
:FUNCtion:DISPlay <display>

<display> ::= {{1 | ON} | {0 | OFF}}
```

The :FUNCtion:DISPlay command turns the display of the function on or off. When ON is selected, the function performs as specified using the other FUNCtion commands. When OFF is selected, function is neither calculated nor displayed.

Query Syntax

```
:FUNCtion:DISPlay?
```

The :FUNCtion:DISPlay? query returns whether the function display is on or off.

Return Format

```
<display><NL>

<display> ::= {1 | 0}
```

See Also

- Introduction to :FUNCtion Commands
- :VIEW
- :BLANk
- :STATus

:FUNCtion Commands

:FUNCtion:OFFSet — N

Command Syntax

```
:FUNCtion:OFFSet <offset>
```

```
<offset> ::= the value at center screen in NR3 format.
```

The :FUNCtion:OFFSet command sets the voltage or vertical value represented at center screen for the selected function. The range of legal values is generally +/-10 times the current scale of the selected function, but will vary by function. If you set the offset to a value outside of the legal range, the offset value is automatically set to the nearest legal value.

NOTE The :FUNCtion:OFFset command is equivalent to the :FUNCtion:REFerence command.

Query Syntax

```
:FUNCtion:OFFSet?
```

The :FUNCtion:OFFSet? query outputs the current offset value for the selected function.

Return Format

```
<offset><NL>
```

```
<offset> ::= the value at center screen in NR3 format.
```

See Also

- Introduction to :FUNCtion Commands
- :FUNCtion:RANGe
- :FUNCtion:REFerence
- :FUNCtion:SCALe

:FUNCtion Commands

:FUNCtion:OPERation — N

Command Syntax

```
:FUNCtion:OPERation <operation>
```

```
<operation> ::= {SUBTract | MULTiply | INTegrate | DIFFerentiate | FFT}.
```

The :FUNCtion:OPERation command sets the desired operation for a function. (FFT = Fast Fourier Transform.)

Query Syntax

```
:FUNCtion:OPERation?
```

The :FUNCtion:OPERation? query returns the current operation for the selected function.

Return Format

```
<operation><NL>
```

```
<operation> ::= {SUBT | MULT | INT | DIFF | FFT}.
```

See Also

- Introduction to :FUNCtion Commands

:FUNCtion Commands

:FUNCtion:RANGe — N

Command Syntax

```
:FUNCtion:RANGe <range>
```

```
<range> ::= the full-scale vertical axis value in NR3 format.
```

The :FUNCtion:RANGe command defines the full-scale vertical axis for the selected function.

Query Syntax

```
:FUNCtion:RANGe?
```

The :FUNCtion:RANGe? query returns the current full-scale range value for the selected function.

Return Format

```
<range><NL>
```

```
<range> ::= the full-scale vertical axis value in NR3 format.
```

The range for ADD, SUBT, MULT is 8E-6 to 800E+3.

The range for the INTegrate function is 8E-9 to 400E+3 (depends on sweep speed).

The range for the DIFFerentiate function is 80E-3 to 8.0E12 (depends on sweep speed).

The range for the FFT (Fast Fourier Transform) function is 8 to 800 dBV.

See Also

- Introduction to :FUNCtion Commands
- :FUNCtion:SCALe

:FUNCtion Commands

:FUNCtion:REFerence — N

Command Syntax

```
:FUNCtion:REFerence <level>
```

```
<level> ::= the current reference level in NR3 format.
```

The range of legal values is from -400.0 dBV to +400.0 dBV depending on the current :FUNCtion:RANGe value. If you set the reference level to a value outside of the legal range, it is automatically set to the nearest legal value.

The :FUNCtion:REFerence command is only used when an FFT (Fast Fourier Transform) operation is selected. The :FUNCtion:REFerence command sets the reference level represented by center screen.

NOTE The FUNCtion:REFerence command is equivalent to the :FUNCtion:OFFSet command.

Query Syntax

```
:FUNCtion:REFerence?
```

The :FUNCtion:REFerence? query returns the current reference level in dBV.

Return Format

```
<level><NL>
```

```
<level> ::= the current reference level in NR3 format.
```

See Also

- Introduction to :FUNCtion Commands
- :FUNCtion:OFFSet
- :FUNCtion:RANGe
- :FUNCtion:SCALe

:FUNCtion:SCALe — N

Command Syntax

```
:FUNCtion:SCALe <scale value>[<suffix>]

<scale value> ::= integer in NR1 format

<suffix> ::= {V | dB}
```

The :FUNCtion:SCALe command sets the vertical scale, or units per division, of the selected function. Legal values for the scale depend on the selected function.

Query Syntax

```
:FUNCtion:SCALe?
```

The :FUNCtion:SCALe? query returns the current scale value for the selected function.

Return Format

```
<scale value><NL>

<scale value> ::= integer in NR1 format
```

See Also

- Introduction to :FUNCtion Commands
- :FUNCtion:RANGe

:FUNCtion Commands

:FUNCtion:SOURce — N

Command Syntax

```
:FUNCtion:SOURce <value>

<value> ::= {CHANnel<n> | ADD | SUBTract | MULTiply}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :FUNCtion:SOURce command is only used when an FFT (Fast Fourier Transform), DIFF, or INT operation is selected (see the:FUNCtion:OPERation command for more information about selecting an operation). The :FUNCtion:SOURce command selects the source for function operations. Choose CHANnel<n>, or ADD, SUBT, or MULT to specify the desired source for function DIFFerentiate, INTegrate, and FFT operations specified by the :FUNCtion:OPERation command.

Query Syntax

```
:FUNCtion:SOURce?
```

The :FUNCtion:SOURce? query returns the current source for function operations.

Return Format

```
<value><NL>

<value> ::= {CHAN<n> | ADD | SUBT | MULT}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

See Also

- Introduction to :FUNCtion Commands
- :FUNCtion:OPERation

:FUNCtion Commands

:FUNCtion:SPAN — N

Command Syntax

```
:FUNCtion:SPAN <span>

<span> ::= the current frequency span in NR3 format. Legal values are 1 Hz to 100 GHz.

If you set the frequency span to a value outside of the legal range, the
step size is automatically set to the nearest legal value.
```

The :FUNCtion:SPAN command sets the frequency span of the display (left graticule to right graticule) when FFT (Fast Fourier Transform) is selected.

Query Syntax

```
:FUNCtion:SPAN?
```

The :FUNCtion:SPAN? query returns the current frequency span in Hertz.

NOTE After a *RST (Reset) or :AUToscale command, the values returned by the:FUNCtion:CENTer? and :FUNCtion:SPAN? queries depend on the current :TIMebase:RANGe value. Once you change either the :FUNCtion:CENTer or :FUNCtion:SPAN value, they no longer track the :TIMebase:RANGe value.

Return Format

```
<span><NL>
```

```
<span> ::= the current frequency span in NR3 format. Legal values are 1 Hz to 100 GHz.
```

See Also

- Introduction to :FUNCtion Commands
- :FUNCtion:CENTer
- :TIMebase:RANGe
- :TIMebase:SCALe

:FUNCtion Commands

:FUNCtion:WINDow — N

Command Syntax

```
:FUNCtion:WINDow <window>

<window> ::= {RECTangular | HANNing | FLATtop}
```

- The RECTangular window is useful for transient signals, and signals where there are an integral number of cycles in the time record.
- The HANNing window is useful for frequency resolution and general purpose use. It is good for resolving two frequencies that are close together, or for making frequency measurements. This is the default window.
- The FLATtop window is best for making accurate amplitude measurements of frequency peaks.

The :FUNCtion:WINDow command allows the selection of three different windowing transforms or operations for the FFT (Fast Fourier Transform) function.

The FFT operation assumes that the time record repeats. Unless an integral number of sampled waveform cycles exist in the record, a discontinuity is created between the end of one record and the beginning of the next. This discontinuity introduces additional frequency components about the peaks into the spectrum. This is referred to as leakage. To minimize leakage, windows that approach zero smoothly at the start and end of the record are employed as filters to the FFTs. Each window is useful for certain classes of input signals.

Query Syntax

```
:FUNCtion:WINDow?
```

The :FUNCtion:WINDow? query returns the value of the window selected for the FFT function.

Return Format

```
<window><NL>

<window> ::= {RECT | HANN | FLAT}
```

See Also

- Introduction to :FUNCtion Commands

:HARDcopy Commands

Set and query the selection of hardcopy device and formatting options. See Introduction to :HARDcopy Commands.

Command	Query	Options and Query Returns
:HARDcopy:FACTors {{0 | OFF} | {1 | ON}}	:HARDcopy:FACTors?	{0 | 1}
:HARDcopy:FFEed {{0 | OFF} | {1 | ON}}	:HARDcopy:FFEed?	{0 | 1}
:HARDcopy:FILename <string>	:HARDcopy:FILename?	<string> ::= quoted ASCII string
:HARDcopy:FORMat <format>	:HARDcopy:FORMat?	<format> ::= {BMP[24bit] | BMP8bit | PNG | CSV | ASCiixy | BINary | PRINter0 | PRINter1}
:HARDcopy:IGColors {{0 | OFF} | {1 | ON}}	:HARDcopy:IGColors?	{0 | 1}
:HARDcopy:PALette <palette>	:HARDcopy:PALette?	<palette> ::= {COLor | GRAYscale}
:HARDcopy:PDRiver <driver>	:HARDcopy:PDRiver?	<driver> ::= {AP2Xxx | AP21xx | {AP2560 | AP25} | {DJ350 | DJ35} | DJ6xx | {DJ630 | DJ63} | DJ6Special | DJ6Photo | DJ8Special | DJ8xx | DJ9Vip | DJ9xx | GVIP | {PS100 | PS10} | CLASer | MLASer | POSTscript}

Introduction to :HARDcopy Commands

The HARDcopy subsystem provides commands to set and query the selection of hardcopy device and formatting options such as inclusion of instrument settings (FACTors) and generation of formfeed (FFEed).

Reporting the Setup. Use :HARDcopy? to query setup information for the HARDcopy subsystem.

Return Format. The following is a sample response from the :HARDcopy? query. In this case, the query was issued following the *RST command.

```
:HARD:FORM BMP;FIL 'print_00';PDR POST;FACT 0;FFE 0;IGC 0;PAL COL
```

:HARDcopy Commands

:HARDcopy:FACTors — N

Command Syntax

```
:HARDcopy:FACTors <factors>

<factors> ::= {{OFF | 0} | {ON | 1}}
```

The HARDcopy:FACTors command controls whether the scale factors are output on the hardcopy dump.

Query Syntax

```
:HARDcopy:FACTors?
```

The :HARDcopy:FACTors? query returns a flag indicating whether scale factors are output on the hardcopy.

Return Format

```
<factors><NL>

<factors> ::= {0 | 1}
```

See Also

- Introduction to :HARDcopy Commands

:HARDcopy Commands

:HARDcopy:FFEed — N

Command Syntax

```
:HARDcopy:FFEed <ffeed>

<ffeed> ::= {{OFF | 0} | {ON | 1}}
```

The HARDcopy:FFEed command controls whether a formfeed is output at the end of a hardcopy dump.

ON (or 1) is only valid when PRINter0 or PRINter1 is set as the :HARDcopy:FORMat type.

Query Syntax

```
:HARDcopy:FFEed?
```

The :HARDcopy:FFEed? query returns a flag indicating whether a formfeed is output at the end of the hardcopy dump.

Return Format

```
<ffeed><NL>

<ffeed> ::= {0 | 1}
```

See Also

- Introduction to :HARDcopy Commands

:HARDcopy Commands

:HARDcopy:FILename — N

Command Syntax

```
:HARDcopy:FILename <string>

<string> ::= quoted ASCII string
```

The HARDcopy:FILename command sets the output filename for those print formats whose output is a file.

Query Syntax

```
:HARDcopy:FILename?
```

The :HARDcopy:FILename? query returns the current hardcopy output filename.

Return Format

```
<string><NL>

<string> ::= quoted ASCII string
```

See Also

- Introduction to :HARDcopy Commands
- :HARDcopy:FORMat

:HARDcopy Commands

:HARDcopy:FORMat — N

Command Syntax

```
:HARDcopy:FORMat <format>

<format> ::= {BMP[24bit] | BMP8bit | PNG | CSV | ASCiixy | BINary
| PRINter0 | PRINter1}
```

The HARDcopy:FORMat command sets the hardcopy format type.

PRINter0 and PRINter1 are only valid when printers are connected to the oscilloscope's USB ports. (The first printer connected/identified is PRINter0 and the second is PRINter1.)

Query Syntax

```
:HARDcopy:FORMat?
```

The :HARDcopy:FORMat? query returns the selected hardcopy format type.

Return Format

```
<format><NL>
```

```
<format> ::= {BMP | BMP8 | PNG | CSV | ASC | BIN | PRIN0 | PRIN1}
```

See Also

- Introduction to :HARDcopy Commands

:HARDcopy Commands

:HARDcopy:IGColors — N

Command Syntax

```
:HARDcopy:IGColors <value>
```

```
<value> ::= {{OFF | 0} | {ON | 1}}
```

The HARDcopy:IGColors command controls whether the graticule colors are inverted or not.

Query Syntax

```
:HARDcopy:IGColors?
```

The :HARDcopy:IGColors? query returns a flag indicating whether graticule colors are inverted or not.

Return Format

```
<value><NL>
```

```
<value> ::= {0 | 1}
```

See Also

- Introduction to :HARDcopy Commands

:HARDcopy Commands

:HARDcopy:PALette — N

Command Syntax

```
:HARDcopy:PALette <palette>
```

```
<palette> ::= {COLor | GRAYscale}
```

The HARDcopy:PALette command sets the hardcopy palette color.

Query Syntax

```
:HARDcopy:PALette?
```

The :HARDcopy:PALette? query returns the selected hardcopy palette color.

Return Format

```
<palette><NL>

<palette> ::= {COL | GRAY}
```

See Also

- Introduction to :HARDcopy Commands

:HARDcopy Commands

:HARDcopy:PDRiver — N

Command Syntax

```
:HARDcopy:PDRiver <driver>

<driver> ::= {AP2Xxx | AP21xx | {AP2560 | AP25} | {DJ350 | DJ35} | DJ6xx |
{DJ630 | DJ63} | DJ6Special | DJ6Photo | DJ8Special | DJ8xx | DJ9Vip | DJ9xx |
GVIP | {PS100 | PS10} | CLASer | MLASer | POSTscript}
```

The HARDcopy:PDRiver command sets the hardcopy printer driver used for the selected printer.

If the correct driver for the selected printer can be identified, it will be selected and cannot be changed.

Query Syntax

```
:HARDcopy:PDRiver?
```

The :HARDcopy:PDRiver? query returns the selected hardcopy printer driver.

Return Format

```
<driver><NL>

<driver> ::= {AP2X | AP21 | AP25 | DJ35 | DJ6 | DJ63 | DJ6S | DJ6P | DJ8S |
DJ8 | DJ9V | DJ9 | GVIP | PS10 | CLAS | MLAS | POST}
```

See Also

- Introduction to :HARDcopy Commands
- :HARDcopy:FORMat

Commands by Subsystem

:MARKer Commands

Set and query the settings of X-axis markers (X1 and X2 cursors) and the Y-axis markers (Y1 and Y2 cursors). See Introduction to :MARKer Commands.

Command	Query	Options and Query Returns
:MARKer:MODE <mode>	:MARKer:MODE?	<mode> ::= {OFF | MEASurement | MANual}
:MARKer:X1Position <position> [suffix]	:MARKer:X1Position?	<position> ::= X1 cursor position value in NR3 format [suffix] ::= {s | ms | us | ns | ps | Hz | kHz | MHz} <return_value> ::= X1 cursor position value in NR3 format
:MARKer:X1Y1source <source>	:MARKer:X1Y1source?	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= <source>
:MARKer:X2Position <position> [suffix]	:MARKer:X2Position?	<position> ::= X2 cursor position value in NR3 format [suffix] ::= {s | ms | us | ns | ps | Hz | kHz | MHz} <return_value> ::= X2 cursor position value in NR3 format
:MARKer:X2Y2source <source>	:MARKer:X2Y2source?	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= <source>
n/a	:MARKer:XDELta?	<return_value> ::= X cursors delta value in NR3 format
:MARKer:Y1Position <position> [suffix]	:MARKer:Y1Position?	<position> ::= Y1 cursor position value in NR3 format [suffix] ::= {V | mV | dB} <return_value> ::= Y1 cursor position value in NR3 format
:MARKer:Y2Position <position> [suffix]	:MARKer:Y2Position?	<position> ::= Y2 cursor position value in NR3 format [suffix] ::= {V | mV | dB} <return_value> ::= Y2 cursor position value in NR3 format

n/a	:MARKer:YDELta?	<return_value> ::= Y cursors delta value in NR3 format

Introduction to :MARKer Commands

The MARKer subsystem commands set and query the settings of X-axis markers (X1 and X2 cursors) and the Y-axis markers (Y1 and Y2 cursors). You can set and query the marker mode and source, the position of the X and Y cursors, and query delta X and delta Y cursor values.

Reporting the Setup. Use :MARKer? to query setup information for the MARKer subsystem.

Return Format. The following is a sample response from the :MARKer? query. In this case, the query was issued following a *RST and :MARKer:MODE:MANual command.

```
:MARK:X1Y1 NONE;X2Y2 NONE;MODE OFF
```

:MARKer Commands

:MARKer:MODE — N

Command Syntax

```
:MARKer:MODE <mode>

<mode> ::= {OFF | MEASurement | MANual}
```

The :MARKer:MODE command sets the cursors mode. OFF removes the cursor information from the display. MANual mode enables manual placement of the X and Y cursors. In MEASurement mode the cursors track the most recent measurement.

If the front-panel cursors are off, or are set to the front-panel Hex or Binary mode, setting :MARKer:MODE MANual will put the cursors in the front-panel Normal mode.

Setting the mode to MEASurement sets the marker sources (:MARKer:X1Y1source and ::MARKer:X2Y2source) to the measurement source (:MEASure:SOURce). Setting the measurement source remotely always sets the marker sources.

Query Syntax

```
:MARKer:MODE?
```

The :MARKer:MODE? query returns the current cursors mode.

Return Format

```
<mode><NL>

<mode> ::= {OFF | MEAS | MAN}
```

See Also

- Introduction to :MARKer Commands

- :MARKer:X1Y1source
- :MARKer:X2Y2source
- :MEASure:SOURce

:MARKer Commands

:MARKer:X1Position — N

Command Syntax

```
:MARKer:X1Position <position> [suffix]

<position> ::= X1 cursor position in NR3 format

<suffix> ::= {s | ms | us | ns | ps | Hz | kHz | MHz}
```

The :MARKer:X1Position command sets :MARKer:MODE to MANual, sets the X1 cursor position and moves the X1 cursor to the specified value.

Query Syntax

```
:MARKer:X1Position?
```

The :MARKer:X1Position? query returns the current X1 cursor position. If the front-panel cursors are off an error is returned. This is functionally equivalent to the obsolete :MEASure:TSTArt command/query.

Return Format

```
<position><NL>

<position> ::= X1 cursor position in NR3 format
```

See Also

- Introduction to :MARKer Commands
- :MARKer:MODE
- :MARKer:X2Position
- :MARKer:X1Y1source
- :MARKer:X2Y2source

:MARKer Commands

:MARKer:X1Y1source — N

Command Syntax

```
:MARKer:X1Y1source <source>
```

```
<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MARKer:X1Y1source command sets the source for the cursors. The channel you specify must be enabled for cursors to be displayed. If the channel or function is not on, an error message is issued. Sending a :MARKer:X1Y1source command will put the cursors in the MANual mode (see :MARKer:MODE).

This product does not allow independent settings of the X1Y1 and X2Y2 marker sources. Setting the source for one pair of markers sets the source for the other. If :MARKer:MODE is set to OFF or MANual, setting :MEASure:SOURce to CHANnel<n>, FUNCtion, or MATH will also set :MARKer:X1Y1source and :MARKer:X2Y2source to this value.

NOTE MATH is an alias for FUNCtion. The query will return FUNC if the source is FUNCtion or MATH.

Query Syntax

```
:MARKer:X1Y1source?
```

The :MARKer:X1Y1source? query returns the current source for the cursors. If all channels are off or if :MARKer:MODE is set to OFF, the query returns NONE.

Return Format

```
<source><NL>

<source> ::= {CHAN<n> | FUNC | NONE}
```

See Also

- Introduction to :MARKer Commands
- :MARKer:MODE
- :MARKer:X2Y2source
- :MEASure:SOURce

:MARKer Commands

:MARKer:X2Position — N

Command Syntax

```
:MARKer:X2Position <position> [suffix]

<position> ::= X2 cursor position in NR3 format

<suffix> ::= {s | ms | us | ns | ps | Hz | kHz | MHz}
```

The :MARKer:X2Position command sets :MARKer:MODE to MANual, sets the X2 cursor position and moves the

X2 cursor to the specified value.

Query Syntax

```
:MARKer:X2Position?
```

The :MARKer:X2Position? query returns current X2 cursor position. If the front-panel cursors are off an error is returned. This is functionally equivalent to the obsolete :MEASure:TSTOp command/query.

Return Format

```
<position><NL>

<position> ::= X2 cursor position in NR3 format
```

See Also

- Introduction to :MARKer Commands
- :MARKer:MODE
- :MARKer:X1Position
- :MARKer:X2Y2source

:MARKer Commands

:MARKer:X2Y2source — N

Command Syntax

```
:MARKer:X2Y2source <source>

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MARKer:X2Y2source command sets the source for the cursors. The channel you specify must be enabled for cursors to be displayed. If the channel or function is not on, an error message is issued. Sending a MARKer:X2Y2source command puts the cursors in the MANual mode (see :MARKer:MODE).

This product does not allow independent settings of the X1Y1 and X2Y2 marker sources. Setting the source for one pair of markers sets the source for the other. If :MARKer:MODE is set to OFF or MANual, setting :MEASure:SOURce to CHANnel<n>, FUNCtion, or MATH will also set :MARKer:X1Y1source and :MARKer:X2Y2source to this value.

NOTE MATH is an alias for FUNCtion. The query will return FUNC if the source is FUNCtion or MATH.

Query Syntax

```
:MARKer:X2Y2source?
```

The :MARKer:X2Y2source? query returns the current source for the cursors. If all channels are off or if :MARKer:MODE is set to OFF, the query returns NONE.

Return Format

```
<source><NL>
```

```
<source> ::= {CHAN<n> | FUNC | NONE}
```

See Also

- Introduction to :MARKer Commands
- :MARKer:MODE
- :MARKer:X1Y1source
- :MEASure:SOURce

:MARKer Commands

:MARKer:XDELta — N

Query Syntax

```
:MARKer:XDELta?
```

The MARKer:XDELta? query returns the value difference between the current X1 and X2 cursor positions.

Xdelta = (Value at X2 cursor) - (Value at X1 cursor)

NOTE If the front-panel cursors are off or are set to Binary or Hex Mode, the marker position values are not defined. Make sure to set:MARKer:MODE to MANual to put the cursors in the front-panel Normal mode.

Return Format

```
<value><NL>
```

```
<value> ::= difference value in NR3 format.
```

See Also

- Introduction to :MARKer Commands
- :MARKer:MODE
- :MARKer:X1Position
- :MARKer:X2Position
- :MARKer:X1Y1source
- :MARKer:X2Y2source

:MARKer Commands

:MARKer:Y1Position — N

Command Syntax

```
:MARKer:Y1Position <position> [suffix]

<position> ::= Y1 cursor position in NR3 format

<suffix> ::= {mV | V | dB}
```

The :MARKer:Y1Position command sets :MARKer:MODE to MANual, sets the Y1 cursor position and moves the Y1 cursor to the specified value.

Query Syntax

```
:MARKer:Y1Position?
```

The :MARKer:Y1Position? query returns current Y1 cursor position. If the front-panel cursors are off an error is returned. This is functionally equivalent to the obsolete :MEASure:VSTArt command/query

Return Format

```
<position><NL>

<position> ::= Y1 cursor position in NR3 format
```

See Also

- Introduction to :MARKer Commands
- :MARKer:MODE
- :MARKer:X1Y1source
- :MARKer:X2Y2source
- :MARKer:Y2Position

:MARKer Commands

:MARKer:Y2Position — N

Command Syntax

```
:MARKer:Y2Position <position> [suffix]

<position> :: ::= Y2 cursor position in NR3 format

<suffix> ::= {mV | V | dB}
```

The :MARKer:Y2Position command sets :MARKer:MODE to MANual, sets the Y2 cursor position and moves the Y2 cursor to the specified value.

Query Syntax

```
:MARKer:Y2Position?
```

The :MARKer:Y2Position? query returns current Y2 cursor position. If the front-panel cursors are off an error is returned. This is functionally equivalent to the obsolete :MEASure:VSTOp command/query.

Return Format

```
<position><NL>
```

```
<position> ::= Y2 cursor position in NR3 format
```

See Also

- Introduction to :MARKer Commands
- :MARKer:MODE
- :MARKer:X1Y1source
- :MARKer:X2Y2source
- :MARKer:Y1Position

:MARKer Commands

:MARKer:YDELta — N

Query Syntax

```
:MARKer:YDELta?
```

The :MARKer:YDELta? query returns the value difference between the current Y1 and Y2 cursor positions.

Ydelta = (Value at Y2 cursor) - (Value at Y1 cursor)

NOTE If the front-panel cursors are off or are set to Binary or Hex Mode, the marker position values are not defined. Make sure to set :MARKer:MODE to MANual to put the cursors in the front-panel Normal mode.

Return Format

```
<value><NL>
```

```
<value> ::= difference value in NR3 format
```

See Also

- Introduction to :MARKer Commands
- :MARKer:MODE
- :MARKer:X1Y1source

- :MARKer:X2Y2source
- :MARKer:Y1Position
- :MARKer:Y2Position

Commands by Subsystem

:MEASure Commands

Select automatic measurements to be made and control time markers. See Introduction to :MEASure Commands.

Command	Query	Options and Query Returns
:MEASure:CLEar	n/a	n/a
:MEASure:COUNter [<source>]	:MEASure:COUNter? [<source>]	<source> ::= {CHANnel<n>} for DSO models <source> ::= {CHANnel<n> | DIGital0,..,DIGital15 |} for MSO models <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= counter frequency in Hertz in NR3 format
:MEASure:DEFine DELay, <delay spec>	:MEASure:DEFine? DELay	<delay spec> ::= <edge_spec1>,<edge_spec2> edge_spec1 ::= [<slope>]<occurrence> edge_spec2 ::= [<slope>]<occurrence> <slope> ::= {+ | -} <occurrence> ::= integer

:MEASure:DEFine THResholds, <threshold spec>	:MEASure:DEFine? THResholds	<threshold spec> ::= {STANdard} | {<threshold mode>,<upper>, <middle>,<lower>}
		<threshold mode> ::= {PERCent | ABSolute}

:MEASure:DELay [<source1>] [,<source2>]	:MEASure:DELay? [<source1>] [,<source2>]	<source1,2> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= floating-point number delay time in seconds in NR3 format
:MEASure:DUTYcycle [<source>]	:MEASure:DUTYcycle? [<source>]	<source> ::= {CHANnel<n> | FUNCtion | MATH} for DSO models

		<source> ::= {CHANnel<n> | DIGital0,..,DIGital15 | FUNCtion | MATH} for MSO models <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= ratio of positive pulse width to period in NR3 format
:MEASure:FALLtime [<source>]	:MEASure:FALLtime? [<source>]	<source> ::= {CHANnel<n> | FUNCtion | MATH} for DSO models <source> ::= {CHANnel<n> | DIGital0,..,DIGital15 | FUNCtion | MATH} for MSO models <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= time in seconds between the lower and upper thresholds in NR3 format
:MEASure:FREQuency [<source>]	:MEASure:FREQuency? [<source>]	<source> ::= {CHANnel<n> | FUNCtion | MATH} for DSO models <source> ::= {CHANnel<n> | DIGital0,..,DIGital15 | FUNCtion | MATH} for MSO models <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= frequency in Hertz in NR3 format
:MEASure:NWIDth [<source>]	:MEASure:NWIDth? [<source>]	<source> ::= {CHANnel<n> | FUNCtion | MATH} for DSO models <source> ::= {CHANnel<n> | DIGital0,..,DIGital15 | FUNCtion | MATH} for MSO models <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= negative pulse width in seconds-NR3 format
:MEASure:OVERshoot [<source>]	:MEASure:OVERshoot? [<source>]	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= the percent of the overshoot of the selected waveform in NR3 format
:MEASure:PERiod [<source>]	:MEASure:PERiod? [<source>]	<source> ::= {CHANnel<n> | FUNCtion | MATH} for DSO models <source> ::= {CHANnel<n> | DIGital0,..,DIGital15 | FUNCtion | MATH} for MSO models

		<n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= waveform period in seconds in NR3 format
:MEASure:PHASe [<source1>] [,<source2>]	:MEASure:PHASe? [<source1>] [,<source2>]	<source1,2> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= the phase angle value in degrees in NR3 format
:MEASure:PREShoot [<source>]	:MEASure:PREShoot? [<source>]	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= the percent of preshoot of the selected waveform in NR3 format
:MEASure:PWIDth [<source>]	:MEASure:PWIDth? [<source>]	<source> ::= {CHANnel<n> | FUNCtion | MATH} for DSO models <source> ::= {CHANnel<n> | DIGital0,..,DIGital15 | FUNCtion | MATH} for MSO models <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= width of positive pulse in seconds in NR3 format
:MEASure:RISEtime [<source>]	:MEASure:RISEtime? [<source>]	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= rise time in seconds in NR3 format
:MEASure:SDEViation [<source>]	:MEASure:SDEViation? [<source>]	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= calculated std deviation in NR3 format
:MEASure:SHOW {1 | ON}	:MEASure:SHOW?	{1}
:MEASure:SOURce [<source1>] [,<source2>]	:MEASure:SOURce?	<source1,2> ::= {CHANnel<n> | FUNCtion | MATH} for DSO models <source1,2> ::= {CHANnel<n> | DIGital0,..,DIGital15 | FUNCtion | MATH} for MSO models <n> ::= 1-2 or 1-4 in NR1 format

		<return_value> ::= {<source> | NONE}
n/a	:MEASure:TEDGe? <slope><occurrence> [,<source>]	<slope> ::= direction of the waveform <occurrence> ::= the transition to be reported <source> ::= {CHANnel<n> | FUNCtion | MATH} for DSO models <source> ::= {CHANnel<n> | DIGital0,..,DIGital15 | FUNCtion | MATH} for MSO models <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= time in seconds of the specified transition
n/a	:MEASure:TVALue? <value>, [<slope>]<occurrence> [,<source>]	<value> ::= voltage level that the waveform must cross. <slope> ::= direction of the waveform when <value> is crossed. <occurrence> ::= transitions reported. <return_value> ::= time in seconds of specified voltage crossing in NR3 format <source> ::= {CHANnel<n> | FUNCtion | MATH} for DSO models <source> ::= {CHANnel<n> | DIGital0,..,DIGital15 | FUNCtion | MATH} for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:MEASure:VAMPlitude [<source>]	:MEASure:VAMPlitude? [<source>]	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= the amplitude of the selected waveform in volts in NR3 format
:MEASure:VAVerage [<source>]	:MEASure:VAVerage? [<source>]	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= calculated average voltage in NR3 format
:MEASure:VBASe [<source>]	:MEASure:VBASe? [<source>]	<source> ::= {CHANnel<n> | FUNCtion | MATH}

		<n> ::= 1-2 or 1-4 in NR1 format <base_voltage> ::= voltage at the base of the selected waveform in NR3 format
:MEASure:VMAX [<source>]	:MEASure:VMAX? [<source>]	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= maximum voltage of the selected waveform in NR3 format
:MEASure:VMIN [<source>]	:MEASure:VMIN? [<source>]	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= minimum voltage of the selected waveform in NR3 format
:MEASure:VPP [<source>]	:MEASure:VPP? [<source>]	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= voltage peak-to-peak of the selected waveform in NR3 format
:MEASure:VRMS [<source>]	:MEASure:VRMS? [<source>]	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= calculated dc RMS voltage in NR3 format
n/a	:MEASure:VTIMe? <vtime> [,<source>]	<vtime> ::= displayed time from trigger in seconds in NR3 format <return_value> ::= voltage at the specified time in NR3 format <source> ::= {CHANnel<n> | FUNCtion | MATH} for DSO models <source> ::= {CHANnel<n> | DIGital0,..,DIGital15 | FUNCtion | MATH} for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:MEASure:VTOP [<source>]	:MEASure:VTOP? [<source>]	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= voltage at the top

		of the waveform in NR3 format
:MEASure:XMAX [<source>]	:MEASure:XMAX? [<source>]	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= horizontal value of the maximum in NR3 format
:MEASure:XMIN [<source>]	:MEASure:XMIN? [<source>]	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= horizontal value of the maximum in NR3 format

Introduction to :MEASure Commands

The commands in the MEASure subsystem are used to make parametric measurements on displayed waveforms.

Measurement Setup. To make a measurement, the portion of the waveform required for that measurement must be displayed on the oscilloscope screen.

Measurement Type	Portion of waveform that must be displayed
period, duty cycle, or frequency	at least one complete cycle
pulse width	the entire pulse
rise time	rising edge, top and bottom of pulse
fall time	falling edge, top and bottom of pulse

Measurement Error. If a measurement cannot be made (typically because the proper portion of the waveform is not displayed), the value +9.9E+37 is returned for that measurement.

Making Measurements. If more than one waveform, edge, or pulse is displayed, time measurements are made on the portion of the displayed waveform closest to the trigger reference (left, center, or right).

When making measurements in the delayed time base mode (:TIMebase:MODE WINDow), the oscilloscope will attempt to make the measurement inside the delayed sweep window. If the measurement is an average and there are not three edges, the oscilloscope will revert to the mode of making the measurement at the start of the main sweep.

When the command form is used, the measurement result is displayed on the instrument. When the query form of these measurements is used, the measurement is made one time, and the measurement result is returned over the bus.

Measurements are made on the displayed waveforms specified by the :MEASure:SOURce command. The MATH source is an alias for the FUNCtion source.

Not all measurements are available on the digital channels or FFT (Fast Fourier Transform).

Reporting the Setup. Use the :MEASure? query to obtain setup information for the MEASure subsystem.

(Currently, this is only :MEASure:SOURce.)

Return Format. The following is a sample response from the :MEASure? query. In this case, the query was issued following a *RST command.

```
:MEAS:SOUR CHAN1,NONE
```

:MEASure Commands

:MEASure:CLEar — N

Command Syntax

```
:MEASure:CLEar
```

This command clears all selected measurements and markers from the screen.

See Also

- Introduction to :MEASure Commands

:MEASure Commands

:MEASure:COUNter — N

Command Syntax

```
:MEASure:COUNter [<source>]

<source> ::= {<digital channels> | CHANnel<n>}

<digital channels> ::= DIGital0,..,DIGital15 for the MSO models

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:COUNter command installs a screen measurement and starts a counter measurement. If the optional source parameter is specified, the current source is modified. Any channel except Math may be selected for the source.

The counter measurement counts trigger level crossings at the selected trigger slope and displays the results in Hz. The gate time for the measurement is automatically adjusted to be 100 ms or twice the current time window, whichever is longer, up to 1 second. The counter measurement can measure frequencies up to 125 MHz. The minimum frequency supported is 1/(2 X gate time).

The Y cursor shows the the edge threshold level used in the measurement.

Only one counter measurement may be displayed at a time.

This command is not available if the source is MATH.

NOTE

Query Syntax

```
:MEASure:COUNter? [<source>]
```

The :MEASure:COUNter? query measures and outputs the counter frequency of the specified source.

NOTE The :MEASure:COUNter? query times out if the counter measurement is installed on the front panel. Use :MEASure:CLEar to remove the front-panel measurement before executing the :MEASure:COUNter? query.

Return Format

```
<source><NL>

<source> ::= count in Hertz in NR3 format
```

See Also

- Introduction to :MEASure Commands
- :MEASure:SOURce
- :MEASure:FREQuency

:MEASure Commands

:MEASure:DEFine — N

Command Syntax

```
:MEASure:DEFine <meas_spec>

<meas_spec> ::= {DELay | THResholds}
```

The :MEASure:DEFine command sets up the definition for measurements by specifying the delta time or threshold values. Changing these values may affect the results of other measure commands. The table below identifies which measurement results that can be affected by redefining the DELay specification or the THResholds values. For example, changing the THResholds definition from the default 10, 50, and 90% values may change the returned measurement result.

MEASure Command	DELay	THResholds
DUTYcycle		x
DELay	x	x
FALLtime		x
FREQuency		x
NWIDth		x

OVERshoot	x
PERiod	x
PHASe	x
PREShoot	x
PWIDth	x
RISetime	x
VAVerage	x
VRMS	x

:MEASure:DEFine DELay Command Syntax

```
:MEASure:DEFine DELay,<delay spec>

<delay spec> ::= <edge_spec1>,<edge_spec2>

<edge_spec1> ::= [<slope>]<occurrence>

<edge_spec2> ::= [<slope>]<occurrence>

<slope> ::= {+ | -}

<occurrence> ::= integer
```

This command defines the behavior of the :MEASure:DELay? query by specifying the start and stop edge to be used. <edge_spec1> specifies the slope and edge number on source1. <edge_spec2> specifies the slope and edge number on source2. The measurement is taken as:

delay = t(<edge_spec2>) - t(<edge_spec1>)

NOTE The :MEASure:DELay command and the front-panel delay measurement use an auto-edge selection method to determine the actual edge used for the measurement. The :MEASure:DEFine command has no effect on these delay measurements. The edges specified by the :MEASure:DEFine command only define the edges used by the :MEASure:DELay? query.

:MEASure:DEFine THResholds Command Syntax

```
:MEASure:DEFine THResholds,<threshold spec>

<threshold spec> ::= {STANdard} | {<threshold mode>,<upper>,<middle>,<lower>}

<threshold mode> ::= {PERCent | ABSolute}
```

for <threshold mode> = PERCent:

```
<upper>, <middle>, <lower> ::= A number specifying the upper, middle, and
lower threshold percentage values between Vbase and Vtop in NR3 format.
```

for <threshold mode> = ABSolute:

```
<upper>, <middle>, <lower> ::= A number specifying the upper, middle, and
```

```
lower threshold absolute values in NR3 format.
```

- STANdard threshold specification sets the lower, middle, and upper measurement thresholds to 10, 50, and 90% values between Vbase and Vtop.
- Threshold mode PERCent sets the measurement thresholds to any user-defined percentages between 5 and 95% of values between Vbase and Vtop.
- Threshold mode ABSolute sets the measurement thresholds to absolute values. ABSolute thresholds are dependent on channel scaling (:CHANnel<n>:RANGe or :CHANnel<n>:SCALe), probe attenuation (:CHANnel<n>:PROBe), and probe units (:CHANnel<n>:UNITs). Always set these values first before setting ABSolute thresholds.

Query Syntax

```
:MEASure:DEFine? <meas_spec>

<meas_spec> ::= {DELay | THResholds}
```

The :MEASure:DEFine? query returns the current edge specification for the delay measurements setup or the current specification for the thresholds setup.

Return Format

for <meas_spec> = DELay:

```
{ <edge_spec1> | <edge_spec2> | <edge_spec1>,<edge_spec2>} <NL>
```

for <meas_spec> = THResholds and <threshold mode> = PERCent:

```
THR,PERC,<upper>,<middle>,<lower><NL>

<upper>, <middle>, <lower> ::= A number specifying the upper, middle, and
lower threshold percentage values between Vbase and Vtop in NR3 format.
```

for<meas_spec> = THResholds and <threshold mode> = ABSolute:

```
THR,ABS,<upper>,<middle>,<lower><NL>

<upper>, <middle>, <lower> ::= A number specifying the upper, middle, and
lower threshold voltages in NR3 format.
```

for <threshold spec> = STANdard:

```
THR,PERC,+90.0,+50.0,+10.0
```

See Also

- Introduction to :MEASure Commands
- :MEASure:DELay
- :MEASure:SOURce

:MEASure Commands

:MEASure:DELay — N

Command Syntax

```
:MEASure:DELay [<source1>][,<source2>]
```

```
<source1>, <source2> ::= {CHANnel<n> | FUNCtion | MATH}
```

```
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
```

```
<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:DELay command places the instrument in the continuous measurement mode and starts a delay measurement.

The measurement is taken as:

delay = t(<edge spec 2>) - t(<edge spec 1>)

where the <edge spec> definitions are set by the :MEASure:DEFine command

NOTE The :MEASure:DELay command and the front-panel delay measurement differ from the :MEASure:DELay? query. The delay command or front-panel measurement run the delay measurement in auto-edge select mode. In this mode, the user may select the edge polarity, but the instrument will select the edges to use to make the best possible delay measurement. The source1 edge chosen will be the edge that meets the polarity specified and is closest to the trigger reference point. The source2 edge selected will be that edge of the specified polarity that gives the first of the following criteria:

- The smallest positive delay value that is less than source1 period.
- The smallest negative delay that is less than source1 period.
- The smallest absolute value of delay.

The :MEASure:DELay? query will make the measurement using the edges specified by the :MEASure:DEFine command.

Query Syntax

```
:MEASure:DELay? [<source1>][,<source2>]
```

The :MEASure:DELay? query measures and returns the delay between source1 and source2. The delay measurement is made from the user-defined slope and edge count of the signal connected to source1, to the defined slope and edge count of the signal connected to source2. Delay measurement slope and edge parameters are selected using the :MEASure:DEFine command.

Return Format

```
<value><NL>
```

```
<value> ::= floating-point number delay time in seconds in NR3 format
```

See Also

- Introduction to :MEASure Commands
- :MEASure:DEFine
- :MEASure:PHASe

:MEASure:DUTYcycle — C

Command Syntax

```
:MEASure:DUTYcycle [<source>]

<source> ::= {<digital channels> | CHANnel<n> | FUNCtion | MATH}

<digital channels> ::= DIGital0,..,DIGital15 for the MSO models

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:DUTYcycle command installs a screen measurement and starts a duty cycle measurement on the current :MEASure:SOURce. If the optional source parameter is specified, the current source is modified.

NOTE The signal must be displayed to make the measurement. This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:DUTYcycle? [<source>]
```

The :MEASure:DUTYcycle? query measures and outputs the duty cycle of the signal specified by the :MEASure:SOURce command. The value returned for the duty cycle is the ratio of the positive pulse width to the period. The positive pulse width and the period of the specified signal are measured, then the duty cycle is calculated with the following formula:

duty cycle = (+pulse width/period)*100

Return Format

```
<value><NL>

<value> ::= ratio of positive pulse width to period inNR3 format
```

See Also

- Introduction to :MEASure Commands
- :MEASure:PERiod
- :MEASure:PWIDth
- :MEASure:SOURce

Example Code

- Example Code

:MEASure Commands

:MEASure:FALLtime — C

Command Syntax

```
:MEASure:FALLtime [<source>]

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:FALLtime command installs a screen measurement and starts a fall-time measurement. For highest measurement accuracy, set the sweep speed as fast as possible, while leaving the falling edge of the waveform on the display. If the optional source parameter is specified, the current source is modified.

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:FALLtime? [<source>]
```

The :MEASure:FALLtime? query measures and outputs the fall time of the displayed falling (negative-going) edge closest to the trigger reference. The fall time is determined by measuring the time at the upper threshold of the falling edge, then measuring the time at the lower threshold of the falling edge, and calculating the fall time with the following formula:

> fall time = time at lower threshold - time at upper threshold

Return Format

```
<value><NL>

<value> ::= time in seconds between the lower threshold and upper threshold in NR3 format
```

See Also

- Introduction to :MEASure Commands
- :MEASure:RISetime
- :MEASure:SOURce

:MEASure Commands

:MEASure:FREQuency — C

Command Syntax

```
:MEASure:FREQuency [<source>]
```

```
<source> ::= {<digital channels> | CHANnel<n> | FUNCtion | MATH}
```

```
<digital channels> ::= DIGital0,..,DIGital15 for the MSO models
```

```
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
```

```
<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:FREQuency command installs a screen measurement and starts a frequency measurement. If the optional source parameter is specified, the current source is modified.

IF the edge on the screen closest to the trigger reference is rising:

THEN frequency = 1/(time at trailing rising edge - time at leading rising edge)

ELSE frequency = 1/(time at trailing falling edge - time at leading falling edge)

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:FREQuency? [<source>]
```

The :MEASure:FREQuency? query measures and outputs the frequency of the cycle on the screen closest to the trigger reference.

Return Format

```
<source><NL>
```

```
<source> ::= frequency in Hertz in NR3 format
```

See Also

- Introduction to :MEASure Commands
- :MEASure:SOURce
- :MEASure:PERiod

Example Code

- Example Code

:MEASure:NWIDth — C

Command Syntax

```
:MEASure:NWIDth [<source>]

<source> ::= {<digital channels> | CHANnel<n> | FUNCtion | MATH}

<digital channels> ::= DIGital0,..,DIGital15 for the MSO models

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:NWIDth command installs a screen measurement and starts a negative pulse width measurement. If the optional source parameter is specified, the current source is modified.

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:NWIDth? [<source>]
```

The :MEASure:NWIDth? query measures and outputs the width of the negative pulse on the screen closest to the trigger reference using the midpoint between the upper and lower thresholds.

FOR the negative pulse closest to the trigger point:

> width = (time at trailing rising edge - time at leading falling edge)

Return Format

```
<value><NL>

<value> ::= negative pulse width in seconds inNR3 format
```

See Also

- Introduction to :MEASure Commands
- :MEASure:SOURce
- :MEASure:PWIDth
- :MEASure:PERiod

:MEASure Commands

:MEASure:OVERshoot — C

Command Syntax

```
:MEASure:OVERshoot [<source>]

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:OVERshoot command installs a screen measurement and starts an overshoot measurement. If the optional source parameter is specified, the current source is modified.

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:OVERshoot? [<source>]
```

The :MEASure:OVERshoot? query measures and returns the overshoot of the edge closest to the trigger reference, displayed on the screen. The method used to determine overshoot is to make three different vertical value measurements: Vtop, Vbase, and either Vmax or Vmin, depending on whether the edge is rising or falling.

For a rising edge:

overshoot = ((Vmax-Vtop) / (Vtop-Vbase)) x 100

For a falling edge:

overshoot = ((Vbase-Vmin) / (Vtop-Vbase)) x 100

Vtop and Vbase are taken from the normal histogram of all waveform vertical values. The extremum of Vmax or Vmin is taken from the waveform interval right after the chosen edge, halfway to the next edge. This more restricted definition is used instead of the normal one, because it is conceivable that a signal may have more preshoot than overshoot, and the normal extremum would then be dominated by the preshoot of the following edge.

Return Format

```
<overshoot><NL>

<overshoot>::= the percent of the overshoot of the selected waveform in NR3 format
```

See Also

- Introduction to :MEASure Commands
- :MEASure:PREShoot
- :MEASure:SOURce
- :MEASure:VMAX
- :MEASure:VTOP
- :MEASure:VBASe

- :MEASure:VMIN

:MEASure:PERiod — C

Command Syntax

```
:MEASure:PERiod [<source>]

<source> ::= {<digital channels> | CHANnel<n> | FUNCtion | MATH}

<digital channels> ::= DIGital0,..,DIGital15 for the MSO models

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:PERiod command installs a screen measurement and starts the period measurement. If the optional source parameter is specified, the current source is modified.

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:PERiod? [<source>]
```

The :MEASure:PERiod? query measures and outputs the period of the cycle closest to the trigger reference on the screen. The period is measured at the midpoint of the upper and lower thresholds.

IF the edge closest to the trigger reference on screen is rising:

THEN period = (time at trailing rising edge - time at leading rising edge)

ELSE period = (time at trailing falling edge - time at leading falling edge)

Return Format

```
<value><NL>

<value> ::= waveform period in seconds in NR3 format
```

See Also

- Introduction to :MEASure Commands
- :MEASure:SOURce
- :MEASure:NWIDth
- :MEASure:PWIDth

- :MEASure:FREQuency

Example Code

- Example Code

:MEASure:PHASe — N

Command Syntax

```
:MEASure:PHASe [<source1>][,<source2>]
```

```
<source1>, <source2> ::= {CHANnel<n> | FUNCtion | MATH}
```

```
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
```

```
<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:PHASe command places the instrument in the continuous measurement mode and starts a phase measurement.

Query Syntax

```
:MEASure:PHASe? [<source1>][,<source2>]
```

The :MEASure:PHASe? query measures and returns the phase between the specified sources.

A phase measurement is a combination of the period and delay measurements. First, the period is measured on source1. Then the delay is measured between source1 and source2. The edges used for delay are the source1 rising edge used for the period measurement closest to the horizontal reference and the rising edge on source 2. See :MEASure:DELay for more detail on selecting the 2nd edge.

The phase is calculated as follows:

> phase = (delay / period of input 1) x 360

Return Format

```
<value><NL>
```

```
<value> ::= the phase angle value in degrees in NR3 format
```

See Also

- Introduction to :MEASure Commands
- :MEASure:DELay
- :MEASure:PERiod
- :MEASure:SOURce

:MEASure:PREShoot — C

Command Syntax

```
:MEASure:PREShoot [<source>]

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:PREShoot command installs a screen measurement and starts a preshoot measurement. If the optional source parameter is specified, the current source is modified.

Query Syntax

```
:MEASure:PREShoot? [<source>]
```

The :MEASure:PREShoot? query measures and returns the preshoot of the edge closest to the trigger, displayed on the screen. The method used to determine preshoot is to make three different vertical value measurements: Vtop, Vbase, and either Vmin or Vmax, depending on whether the edge is rising or falling.

For a rising edge:

> preshoot = ((Vmin-Vbase) / (Vtop-Vbase)) x 100

For a falling edge:

> preshoot = ((Vmax-Vtop) / (Vtop-Vbase)) x 100

Vtop and Vbase are taken from the normal histogram of all waveform vertical values. The extremum of Vmax or Vmin is taken from the waveform interval right before the chosen edge, halfway back to the previous edge. This more restricted definition is used instead of the normal one, because it is likely that a signal may have more overshoot than preshoot, and the normal extremum would then be dominated by the overshoot of the preceding edge.

Return Format

```
<value><NL>

<value> ::= the percent of preshoot of the selected waveform in NR3 format
```

See Also

- Introduction to :MEASure Commands
- :MEASure:SOURce
- :MEASure:VMIN
- :MEASure:VMAX

- :MEASure:VTOP
- :MEASure:VBASe

:MEASure Commands

:MEASure:PWIDth — C

Command Syntax

```
:MEASure:PWIDth [<source>]

<source> ::= {<digital channels> | CHANnel<n> | FUNCtion | MATH}

<digital channels> ::= DIGital0,..,DIGital15 for the MSO models

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:PWIDth command installs a screen measurement and starts the positive pulse width measurement. If the optional source parameter is specified, the current source is modified.

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:PWIDth? [<source>]
```

The :MEASure:PWIDth? query measures and outputs the width of the displayed positive pulse closest to the trigger reference. Pulse width is measured at the midpoint of the upper and lower thresholds.

IF the edge on the screen closest to the trigger is falling:

THEN width = (time at trailing falling edge - time at leading rising edge)

ELSE width = (time at leading falling edge - time at leading rising edge)

Return Format

```
<value><NL>

<value> ::= width of positive pulse in seconds in NR3 format
```

See Also

- Introduction to :MEASure Commands
- :MEASure:SOURce
- :MEASure:NWIDth

- :MEASure:PERiod

:MEASure Commands

:MEASure:RISetime — C

Command Syntax

```
:MEASure: RISetime [<source>]

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:RISetime command installs a screen measurement and starts a rise-time measurement. If the optional source parameter is specified, the current source is modified.

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure: RISetime? [<source>]
```

The :MEASure:RISetime? query measures and outputs the rise time of the displayed rising (positive-going) edge closest to the trigger reference. For maximum measurement accuracy, set the sweep speed as fast as possible while leaving the leading edge of the waveform on the display. The rise time is determined by measuring the time at the lower threshold of the rising edge and the time at the upper threshold of the rising edge, then calculating the rise time with the following formula:

> rise time = time at upper threshold - time at lower threshold

Return Format

```
<value><NL>

<value> ::= rise time in seconds in NR3 format
```

See Also

- Introduction to :MEASure Commands
- :MEASure:SOURce
- :MEASure:FALLtime

:MEASure Commands

:MEASure:SDEViation — N

Command Syntax

```
:MEASure:SDEViation [<source>]

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:SDEViation command installs a screen measurement and starts std deviation measurement. If the optional source parameter is specified, the current source is modified.

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:SDEViation? [<source>]
```

The :MEASure:SDEViation? query measures and outputs the std deviation of the selected waveform. The oscilloscope computes the std deviation on all displayed data points.

Return Format

```
<value><NL>

<value> ::= calculated std deviation value in NR3 format
```

See Also

- Introduction to :MEASure Commands
- :MEASure:SOURce

:MEASure Commands

:MEASure:SHOW — N

Command Syntax

```
:MEASure:SHOW <show>

<show> ::= {1 | ON}
```

The :MEASure:SHOW command enables markers for tracking measurements on the display. This feature is always on in the 6000 Series oscilloscopes.

Query Syntax

```
:MEASure:SHOW?
```

The :MEASure:SHOW? query returns the current state of the markers.

Return Format

```
<show><NL>

<show> ::= 1
```

See Also

- Introduction to :MEASure Commands

:MEASure Commands

:MEASure:SOURce — C

Command Syntax

```
:MEASure:SOURce <source1>[,<source2>]

<source1>,<source2> ::= {<digital channels> | CHANnel<n> | FUNCtion | MATH}

<digital channels> ::= DIGital0,..,DIGital15 for the MSO models

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:SOURce command sets the default sources for measurements. The specified sources are used as the sources for the MEASure subsystem commands if the sources are not explicitly set with the command. If a source is specified for any measurement, the current source is changed to this new value.
If :MARKer:MODE is set to OFF or MANual, setting :MEASure:SOURce to CHANnel<n>, FUNCtion, or MATH will also set :MARKer:X1Y1source to source1 and :MARKer:X2Y2source to source2.

Query Syntax

```
:MEASure:SOURce?
```

The :MEASure:SOURce? query returns the current source selections. If source2 is not specified, the query returns "NONE" for source2. If all channels are off, the query returns "NONE,NONE". Source2 only applies to :MEASure:DELay and :MEASure:PHASe measurements.

NOTE MATH is an alias for FUNCtion. The query will return FUNC if the source is FUNCtion or MATH.

Return Format

```
<source1>,<source2><NL>

<source1>,<source2> ::= {<digital channels> | CHAN<n> | FUNC | NONE}
```

See Also

- Introduction to :MEASure Commands

Example Code

```
' MEASURE - The commands in the MEASURE subsystem are used to make
' measurements on displayed waveforms.
myScope.WriteString ":MEASURE:SOURCE CHANNEL1"   ' Source to measure.
myScope.WriteString ":MEASURE:FREQUENCY?"   ' Query for frequency.
varQueryResult = myScope.ReadNumber   ' Read frequency.
MsgBox "Frequency:" + vbCrLf + FormatNumber(varQueryResult / 1000, 4) + " kHz"
myScope.WriteString ":MEASURE:DUTYCYCLE?"   ' Query for duty cycle.
varQueryResult = myScope.ReadNumber   ' Read duty cycle.
MsgBox "Duty cycle:" + vbCrLf + FormatNumber(varQueryResult, 3) + "%"
myScope.WriteString ":MEASURE:RISETIME?"   ' Query for risetime.
varQueryResult = myScope.ReadNumber   ' Read risetime.
MsgBox "Risetime:" + vbCrLf + FormatNumber(varQueryResult * 1000000, 4) + " us"
myScope.WriteString ":MEASURE:VPP?"   ' Query for Peak to Peak voltage.
varQueryResult = myScope.ReadNumber   ' Read VPP.
MsgBox "Peak to peak voltage:" + vbCrLf + FormatNumber(varQueryResult, 4) + " V"
myScope.WriteString ":MEASURE:VMAX?"   ' Query for Vmax.
varQueryResult = myScope.ReadNumber   ' Read Vmax.
MsgBox "Maximum voltage:" + vbCrLf + FormatNumber(varQueryResult, 4) + " V"
```

Example program from the start: VISA COM Example in Visual Basic

:MEASure Commands

:MEASure:TEDGe — N

Query Syntax

```
:MEASure:TEDGe? <slope><occurrence>[,<source>]
```

```
<slope> ::= direction of the waveform. A rising slope is indicated by a space or
plus sign (+). A falling edge is indicated by a minus sign (-).
```

```
<occurrence> ::= the transition to be reported. If the occurrence number is one,
the first crossing from the left screen edge is reported.  If the number is two,
the second crossing is reported, etc.
```

```
<source> ::= {<digital channels> | CHANnel<n> | FUNCtion | MATH}
```

```
<digital channels> ::= DIGital0,..,DIGital15 for the MSO models
```

```
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
```

```
<n> ::= {1 | 2} for the two channel oscilloscope models
```

When the :MEASure:TEDGe query is sent, the displayed signal is searched for the specified transition. The time interval between the trigger event and this occurrence is returned as the response to the query. The sign of the slope selects a rising (+) or falling (-) edge. If no sign is specified for the slope, it is assumed to be the rising edge.

The magnitude of occurrence defines the occurrence to be reported. For example, +3 returns the time for the third time the waveform crosses the midpoint threshold in the positive direction. Once this crossing is found, the oscilloscope reports the time at that crossing in seconds, with the trigger point (time zero) as the reference.

If the specified crossing cannot be found, the oscilloscope reports +9.9E+37. This value is returned if the waveform does not cross the specified vertical value, or if the waveform does not cross the specified vertical value for the specific number of times in the direction specified.

You can make delay and phase measurements using the MEASure:TEDGe command:

> Delay = time at the nth rising or falling edge of the channel - time at the same edge of another channel
>
> Phase = (delay between channels / period of channel) x 360

For an example of making a delay and phase measurement, see :MEASure:TEDGe Code.

If the optional source parameter is specified, the current source is modified.

NOTE This query is not available if the source is FFT (Fast Fourier Transform).

Return Format

`<value><NL>`

`<value> ::= time in seconds of the specified transition in NR3 format`

:MEASure:TEDGe Code

```
' Make a delay measurement between channel 1 and 2.
Dim dblChan1Edge1 As Double
Dim dblChan2Edge1 As Double
Dim dblChan1Edge2 As Double
Dim dblDelay As Double
Dim dblPeriod As Double
Dim dblPhase As Double

myScope.WriteString ":MEASURE:TEDGE? +1, CHAN1"   ' Query time at 1st rising edge on ch1.
dblChan1Edge1 = myScope.ReadNumber   ' Read time at edge 1 on ch 1.
myScope.WriteString ":MEASURE:TEDGE? +1, CHAN2"   ' Query time at 1st rising edge on ch2.
dblChan2Edge1 = myScope.ReadNumber   ' Read time at edge 1 on ch 2.
dblDelay = dblChan2Edge1 - dblChan1Edge1   ' Calculate delay time between ch1 and ch2.
MsgBox "Delay = " + vbCrLf + CStr(dblDelay)   ' Write calculated delay time to screen.

' Make a phase difference measurement between channel 1 and 2.
myScope.WriteString ":MEASURE:TEDGE? +2, CHAN1"   ' Query time at 1st rising edge on ch1.
dblChan1Edge2 = myScope.ReadNumber   ' Read time at edge 2 on ch 1.
dblPeriod = dblChan1Edge2 - dblChan1Edge1   ' Calculate period of ch 1.
dblPhase = (dblDelay / dblPeriod) * 360 ' Calculate phase difference between ch1 and ch2.
MsgBox "Phase = " + vbCrLf + CStr(dblPhase)
```

Example program from the start: VISA COM Example in Visual Basic

See Also

- Introduction to :MEASure Commands

- :MEASure:TVALue
- :MEASure:VTIMe

:MEASure Commands

:MEASure:TVALue — C

Query Syntax

```
:MEASure:TVALue? <value>, [<slope>]<occurrence>[,<source>]

<value> ::= the vertical value that the waveform must cross.  The value can be
volts or a math function value such as dB, Vs, or V/s

<slope> ::= direction of the waveform. A rising slope is indicated by a plus
sign (+). A falling edge is indicated by a minus sign (-).

<occurrence> ::= the transition to be reported. If the occurrence number is
one, the first crossing is reported.  If the number is two, the second crossing
is reported, etc.

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

When the :MEASure:TVALue? query is sent, the displayed signal is searched for the specified value level and transition. The time interval between the trigger event and this defined occurrence is returned as the response to the query.

The specified value can be negative or positive. To specify a negative value, use a minus sign (-). The sign of the slope selects a rising (+) or falling (-) edge. If no sign is specified for the slope, it is assumed to be the rising edge.

The magnitude of the occurrence defines the occurrence to be reported. For example, +3 returns the time for the third time the waveform crosses the specified value level in the positive direction. Once this value crossing is found, the oscilloscope reports the time at that crossing in seconds, with the trigger point (time zero) as the reference.

If the specified crossing cannot be found, the oscilloscope reports +9.9E+37. This value is returned if the waveform does not cross the specified value, or if the waveform does not cross the specified value for the specified number of times in the direction specified.

If the optional source parameter is specified, the current source is modified.

NOTE This query is not available if the source is FFT (Fast Fourier Transform).

Return Format

```
<value><NL>
```

```
<value> ::= time in seconds of the specified value crossing in NR3 format
```

See Also

- Introduction to :MEASure Commands
- :MEASure:TEDGe
- :MEASure:VTIMe

:MEASure Commands

:MEASure:VAMPlitude — C

Command Syntax

```
:MEASure:VAMPlitude [<source>]

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:VAMPlitude command installs a screen measurement and starts a vertical amplitude measurement. If the optional source parameter is specified, the current source is modified.

Query Syntax

```
:MEASure:VAMPlitude? [<source>]
```

The :MEASure:VAMPlitude? query measures and returns the vertical amplitude of the waveform. To determine the amplitude, the instrument measures Vtop and Vbase, then calculates the amplitude as follows:

> vertical amplitude = Vtop - Vbase

Return Format

```
<value><NL>

<value> ::= the amplitude of the selected waveform in NR3 format
```

See Also

- Introduction to :MEASure Commands
- :MEASure:SOURce
- :MEASure:VBASe
- :MEASure:VTOP
- :MEASure:VPP

:MEASure Commands

:MEASure:VAVerage — C

Command Syntax

```
:MEASure:VAVerage [<source>]

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:VAVerage command installs a screen measurement and starts an average value measurement. If the optional source parameter is specified, the current source is modified.

Query Syntax

```
:MEASure:VAVerage? [<source>]
```

The :MEASure:VAVerage? query returns the average value of an integral number of periods of the signal. If at least three edges are not present, the oscilloscope averages all data points.

Return Format

```
<value><NL>

<value> ::= calculated average value in NR3 format
```

See Also

- Introduction to :MEASure Commands
- :MEASure:SOURce

:MEASure Commands

:MEASure:VBASe — C

Command Syntax

```
:MEASure:VBASe [<source>]

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:VBASe command installs a screen measurement and starts a waveform base value measurement. If the optional source parameter is specified, the current source is modified.

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:VBASe? [<source>]
```

The :MEASure:VBASe? query returns the vertical value at the base of the waveform. The base value of a pulse is normally not the same as the minimum value.

Return Format

```
<base_voltage><NL>

<base_voltage> ::= value at the base of the selected waveform in NR3 format
```

See Also

- Introduction to :MEASure Commands
- :MEASure:SOURce
- :MEASure:VTOP
- :MEASure:VAMPlitude
- :MEASure:VMIN

:MEASure Commands

:MEASure:VMAX — C

Command Syntax

```
:MEASure:VMAX [<source>]

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:VMAX command installs a screen measurement and starts a maximum vertical value measurement. If the optional source parameter is specified, the current source is modified.

Query Syntax

```
:MEASure:VMAX? [<source>]
```

The :MEASure:VMAX? query measures and outputs the maximum vertical value present on the selected waveform.

Return Format

```
<value><NL>
```

```
<value> ::= maximum vertical value of the selected waveform in NR3 format
```

See Also

- Introduction to :MEASure Commands
- :MEASure:SOURce
- :MEASure:VMIN
- :MEASure:VPP
- :MEASure:VTOP

:MEASure Commands

:MEASure:VMIN — C

Command Syntax

```
:MEASure:VMIN [<source>]
```

```
<source> ::= {CHANnel<n> | FUNCtion | MATH}
```

```
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
```

```
<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:VMIN command installs a screen measurement and starts a minimum vertical value measurement. If the optional source parameter is specified, the current source is modified.

Query Syntax

```
:MEASure:VMIN? [<source>]
```

The :MEASure:VMIN? query measures and outputs the minimum vertical value present on the selected waveform.

Return Format

```
<value><NL>
```

```
<value> ::= minimum vertical value of the selected waveform in NR3 format
```

See Also

- Introduction to :MEASure Commands
- :MEASure:SOURce
- :MEASure:VBASe
- :MEASure:VMAX
- :MEASure:VPP

:MEASure Commands

:MEASure:VPP — C

Command Syntax

```
:MEASure:VPP [<source>]

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:VPP command installs a screen measurement and starts a vertical peak-to-peak measurement. If the optional source parameter is specified, the current source is modified.

Query Syntax

```
:MEASure:VPP? [<source>]
```

The :MEASure:VPP? query measures the maximum and minimum vertical value for the selected source, then calculates the vertical peak-to-peak value and returns that value. The peak-to-peak value (Vpp) is calculated with the following formula:

Vpp = Vmax - Vmin

Vmax and Vmin are the vertical maximum and minimum values present on the selected source.

Return Format

```
<value><NL>

<value> ::= vertical peak to peak value in NR3 format
```

See Also

- Introduction to :MEASure Commands
- :MEASure:SOURce
- :MEASure:VMAX
- :MEASure:VMIN
- :MEASure:VAMPlitude

:MEASure Commands

:MEASure:VRMS — C

Command Syntax

```
:MEASure:VRMS [<source>]

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:VRMS command installs a screen measurement and starts a dc RMS value measurement. If the optional source parameter is specified, the current source is modified.

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:VRMS? [<source>]
```

The :MEASure:VRMS? query measures and outputs the dc RMS value of the selected waveform. The dc RMS value is measured on an integral number of periods of the displayed signal. If at least three edges are not present, the oscilloscope computes the RMS value on all displayed data points.

Return Format

```
<value><NL>

<value> ::= calculated dc RMS value in NR3 format
```

See Also

- Introduction to :MEASure Commands
- :MEASure:SOURce

:MEASure Commands

:MEASure:VTIMe — N

Query Syntax

```
:MEASure:VTIMe? <vtime_argument>[,<source>]

<vtime_argument> ::= time from trigger in seconds

<source> ::= {<digital channels> | CHANnel<n> | FUNCtion | MATH}

<digital channels> ::= DIGital0,..,DIGital15 for the MSO models

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:VTIMe? query returns the value at a specified time on the source specified with :MEASure:SOURce. The specified time must be on the screen and is referenced to the trigger event. If the optional source parameter is specified, the current source is modified.

NOTE This query is not available if the source is FFT (Fast Fourier Transform).

Return Format

```
<value><NL>

<value> ::= value at the specified time in NR3 format
```

See Also

- Introduction to :MEASure Commands
- :MEASure:SOURce
- :MEASure:TEDGe
- :MEASure:TVALue

:MEASure Commands

:MEASure:VTOP — C

Command Syntax

```
:MEASure:VTOP [<source>]

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:VTOP command installs a screen measurement and starts a waveform top value measurement.

NOTE This query is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:VTOP? [<source>]
```

The :MEASure:VTOP? query returns the vertical value at the top of the waveform. The top value of the pulse is normally not the same as the maximum value.

Return Format

```
<value><NL>
```

```
<value> ::= vertical value at the top of the waveform in NR3 format
```

See Also

- Introduction to :MEASure Commands
- :MEASure:SOURce
- :MEASure:VMAX
- :MEASure:VAMPlitude
- :MEASure:VBASe

:MEASure Commands

:MEASure:XMAX — N

Command Syntax

```
:MEASure:XMAX [<source>]

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:XMAX command installs a screen measurement and starts an X-at-Max-Y measurement on the selected window. If the optional source parameter is specified, the current source is modified.

NOTE :MEASure:XMAX is an alias for :MEASure:TMAX.

Query Syntax

```
:MEASure:XMAX? [<source>]
```

The :MEASure:XMAX? query measures and returns the horizontal axis value at which the maximum vertical value occurs. If the optional source is specified, the current source is modified. If all channels are off, the query returns 9.9E+37.

Return Format

```
<value><NL>

<value> ::= horizontal value of the maximum in NR3 format
```

See Also

- Introduction to :MEASure Commands
- :MEASure:XMIN

:MEASure Commands

:MEASure:XMIN — N

Command Syntax

```
:MEASure:XMIN [<source>]

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:XMIN command installs a screen measurement and starts an X-at-Min-Y measurement on the selected window. If the optional source parameter is specified, the current source is modified.

NOTE :MEASure:XMIN is an alias for :MEASure:TMIN.

Query Syntax

```
:MEASure:XMIN? [<source>]
```

The :MEASure:XMIN? query measures and returns the horizontal axis value at which the minimum vertical value occurs. If the optional source is specified, the current source is modified. If all channels are off, the query returns 9.9E+37.

Return Format

```
<value><NL>

<value> ::= horizontal value of the minimum in NR3 format
```

See Also

- Introduction to :MEASure Commands
- :MEASure:XMAX

Commands by Subsystem

:POD Commands

Control all oscilloscope functions associated with groups of digital channels. See Introduction to :POD<n> Commands.

Command	Query	Options and Query Returns
:POD<n>:DISPlay {{0 | OFF} | {1 | ON}}	:POD<n>:DISPlay?	{0 | 1}

		<n> ::= 1-2 in NR1 format
:POD<n>:SIZE <value>	:POD<n>:SIZE?	<value> ::= {SMALl | MEDium | LARGe}

:POD<n>:THReshold <type> [suffix]	:POD<n>:THReshold?	<n> ::= 1-2 in NR1 format
		<type> ::= {CMOS | ECL | TTL | <user defined value>}
		<user defined value> ::= value in NR3 format
		[suffix] ::= {V | mV | uV }

Introduction to :POD<n> Commands

<n> ::= {1 | 2}

The POD subsystem commands control the viewing and threshold of groups of digital channels.

POD1 ::= D0-D7

POD2 ::= D8-D15

NOTE These commands are only valid for the MSO models.

Reporting the Setup. Use :POD1? or :POD2? to query setup information for the POD subsystem.

Return Format. The following is a sample response from the :POD1? query. In this case, the query was issued following a *RST command.

```
:POD1:DISP 0;THR +1.40E+00
```

:POD Commands

:POD<n>:DISPlay — N

Command Syntax

```
:POD<n>:DISPlay <display>
```

<display> ::= {{1 | ON} | {0 | OFF}}

<n> ::= An integer, 1 or 2, is attached as a suffix to the command and defines the group of channels that are affected by the command.

POD1 ::= D0-D7

POD2 ::= D8-D15

The :POD<n>:DISPlay command turns displaying of the specified group of channels on or off.

NOTE This command is only valid for the MSO models.

Query Syntax

```
:POD<n>:DISPlay?
```

The :POD<n>:DISPlay? query returns the current display setting of the specified group of channels.

Return Format

```
<display><NL>

<display> ::= {0 | 1}
```

See Also

- Introduction to :POD<n> Commands
- :DIGital<n>:DISPlay
- :CHANnel<n>:DISPlay
- :VIEW
- :BLANk
- :STATus

:POD Commands

:POD<n>:SIZE — N

Command Syntax

```
:POD<n>:SIZE <value>

<n> ::= An integer, 1 or 2, is attached as a suffix to the command and
defines the group of channels that are affected by the command.

POD1 ::= D0-D7

POD2 ::= D8-D15

<value> ::= {SMALl | MEDium | LARGe}
```

The :POD<n>:SIZE command specifies the size of digital channels on the display.

NOTE This command is only valid for the MSO models.

Query Syntax

```
:POD<n>:SIZE?
```

The :POD<n>:SIZE? query returns the size setting for the specified group of channels.

Return Format

```
<size_value><NL>

<size_value> ::= {SMAL | MED | LARG}
```

See Also

- Introduction to :POD<n> Commands
- :DIGital<n>:SIZE
- :DIGital<n>:POSition

:POD Commands

:POD<n>:THReshold — N

Command Syntax

```
:POD<n>:THReshold <type>[<suffix>]

<n> ::= An integer, 1 or 2, is attached as a suffix to the command and
defines the group of channels that are affected by the command.

<type> ::= {CMOS | ECL | TTL | <user defined value>}

<user defined value> ::= -8.00 to +8.00 in NR3 format

<suffix> ::= {V | mV | uV}

POD1 ::= D0-D7

POD2 ::= D8-D15

TTL ::= 1.4V

CMOS ::= 2.5V

ECL ::= -1.3V
```

The :POD<n>:THReshold command sets the threshold for the specified group of channels. The threshold is used for triggering purposes and for displaying the digital data as high (above the threshold) or low (below the threshold).

NOTE This command is only valid for the MSO models.

Query Syntax

`:POD<n>:THReshold?`

The :POD<n>:THReshold? query returns the threshold value for the specified group of channels.

Return Format

`<threshold><NL>`

`<threshold> ::= Floating point number in NR3 format`

See Also

- Introduction to :POD<n> Commands
- :DIGital<n>:THReshold
- :TRIGger[:EDGE]:LEVel

Example Code

```
' THRESHOLD - This command is used to set the voltage threshold for
' the waveforms.  There are three preset values (TTL, CMOS, and ECL)
' and you can also set a user-defined threshold value between
' -8.0 volts and +8.0 volts.
'
' In this example, we set channels 0-7 to CMOS, then set channels
' 8-15 to a user-defined 2.0 volts, and then set the external trigger
' to TTL.  Of course, you only need to set the thresholds for the
' channels you will be using in your program.
myScope.WriteString ":POD1:THRESHOLD CMOS"  ' Set channels 0-7 to CMOS threshold.
myScope.WriteString ":POD2:THRESHOLD 2.0"   ' Set channels 8-15 to 2.0 volts
myScope.WriteString ":TRIG:LEV TTL,EXT"     ' Set external channel to TTL
                                            ' threshold (short form).
```

Example program from the start: VISA COM Example in Visual Basic

Commands by Subsystem

Root (:) Commands

Control many of the basic functions of the oscilloscope and reside at the root level of the command tree. See Introduction to Root (:) Commands.

Command	Query	Options and Query Returns
:ACTivity	:ACTivity?	<return value> ::= <edges>,<levels> <edges> ::= presence of edges (32-bit integer in NR1 format) <levels> ::= logical highs or lows (32-bit integer in NR1 format)
n/a	:AER?	{0 | 1}; an integer in NR1 format
:AUToscale [<source> [,..,<source>]]	n/a	<source> ::= CHANnel<n> for DSO models

		<source> ::= {CHANnel<n> | DIGital0,..,DIGital15 | POD1 | POD2} for MSO models <source> can be repeated up to 5 times <n> ::= 1-2 or 1-4 in NR1 format
:AUToscale:AMODE <value>	:AUToscale:AMODE?	<value> ::= {NORMal | CURRent}}
:AUToscale:CHANnels <value>	:AUToscale:CHANnels?	<value> ::= {ALL | DISPlayed}}
:BLANk [<source>]	n/a	<source> ::= {CHANnel<n>} | FUNCtion | MATH | SBUS} for DSO models <source> ::= {CHANnel<n> | DIGital0,..,DIGital15 | POD{1 | 2} | BUS{1 | 2} | FUNCtion | MATH | SBUS} for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:CDISplay	n/a	n/a
:DIGitize [<source>[,..,<source>]]	n/a	<source> ::= {CHANnel<n> | FUNCtion | MATH | SBUS} for DSO models <source> ::= {CHANnel<n> | DIGital0,..,DIGital15 | POD{1 | 2} | BUS{1 | 2} | FUNCtion | MATH | SBUS} for MSO models <source> can be repeated up to 5 times <n> ::= 1-2 or 1-4 in NR1 format
:HWEenable <n>	:HWEenable?	<n> ::= 16-bit integer in NR1 format
n/a	:HWERregister:CONDition?	<n> ::= 16-bit integer in NR1 format
n/a	:HWERegister[:EVENt]?	<n> ::= 16-bit integer in NR1 format
:MERGe <pixel memory>	n/a	<pixel memory> ::= {PMEMory{0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9}}
:OPEE <n>	:OPEE?	<n> ::= 16-bit integer in NR1 format
n/a	:OPERregister:CONDition?	<n> ::= 16-bit integer in NR1 format
n/a	:OPERegister[:EVENt]?	<n> ::= 16-bit integer in NR1 format
:OVLenable <mask>	:OVLenable?	<mask> ::= 16-bit integer in NR1 format as shown: Bit Weight Input 10 1024 External Trigger Fault 9 512 Channel 4 Fault 8 256 Channel 3 Fault

7	128	Channel 2 Fault
6	64	Channel 1 Fault
4	16	External Trigger OVL
3	8	Channel 4 OVL
2	4	Channel 3 OVL
1	2	Channel 2 OVL
0	1	Channel 1 OVL

n/a	:OVLRegister?	<value> ::= integer in NR1 format. See OVLenable for <value>
:PRINt [<options>]	n/a	<options> ::= [<print option>][,..,<print option>] <print option> ::= {COLor | GRAYscale | PRINter0 | BMP8bit | BMP | PNG | NOFactors | FACTors} <print option> can be repeated up to 5 times.
:RUN	n/a	n/a
n/a	:SERial	<return value> ::= unquoted string containing serial number
:SINGle	n/a	n/a

n/a	:STATus? <display>	{0 | 1}
		<display> ::= {CHANnel<n> | DIGital0,..,DIGital15 | POD{1 | 2} | BUS{1 | 2} | FUNCtion | MATH | SBUS}
		<n> ::= 1-2 or 1-4 in NR1 format

:STOP	n/a	n/a
n/a	:TER?	{0 | 1}
:VIEW <source>	n/a	<source> ::= {CHANnel<n> | PMEMory{0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9} | FUNCtion | MATH | SBUS} for DSO models <source> ::= {CHANnel<n> | DIGital0,..,DIGital15 | PMEMory{0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9} | POD{1 | 2} | BUS{1 | 2} | FUNCtion | MATH | SBUS} for MSO models <n> ::= 1-2 or 1-4 in NR1 format

Introduction to Root (:) Commands

Root level commands control many of the basic operations of the instrument. These commands are always recognized by the parser if they are prefixed with a colon, regardless of current command tree position. After executing a root-level command, the parser is positioned at the root of the command tree.

:ACTivity — N

Command Syntax

```
:ACTivity
```

The :ACTivity command clears the cumulative edge variables for the next activity query.

Query Syntax

```
:ACTivity?
```

The :ACTivity? query returns whether there has been activity (edges) on the digital channels since the last query, and returns the current logic levels.

NOTE Because the :ACTivity? query returns edge activity since the last :ACTivity? query, you must send this query twice before the edge activity result is valid.

Return Format

```
<edges>,<levels><NL>

<edges> ::= presence of edges (16-bit integer in NR1 format).

<levels> ::= logical highs or lows (16-bit integer in NR1 format).

bit 0 ::= DIGital 0

bit 15 ::= DIGital 15
```

NOTE A bit = 0 (zero) in the <edges> result indicates that no edges were detected on that channel (across the specified threshold voltage) since the last query.

A bit = 1 (one) in the <edges> result indicates that edges have been detected on that channel (across the specified threshold voltage) since the last query.

(The threshold voltage must be set appropriately for the logic levels of the signals being probed.)

See Also

- Introduction to Root (:) Commands
- :POD<n>:THReshold
- :DIGital<n>:THReshold

:AER (Arm Event Register) — C

Query Syntax

```
:AER?
```

The AER query reads the Arm Event Register. After the Arm Event Register is read, it is cleared. A "1" indicates the trigger system is in the armed state, ready to accept a trigger.

The Armed Event Register is summarized in the Wait Trig bit of the Operation Status Event Register. A Service Request can be generated when the Wait Trig bit transitions and the appropriate enable bits have been set in the Operation Status Enable Register (OPEE) and the Service Request Enable Register (SRE).

Return Format

```
<value><NL>

<value> ::= {0 | 1}; an integer in NR1 format.
```

See Also

- Introduction to Root (:) Commands
- :OPEE (Operation Status Enable Register)
- :OPERegister:CONDition (Operation Status Condition Register)
- :OPERegister[:EVENt] (Operation Status Event Register)
- *STB (Read Status Byte)
- *SRE (Service Request Enable)

Root (:) Commands

:AUToscale — C

Command Syntax

```
:AUToscale

:AUToscale [<source>[,..,<source>]]

<source> ::= CHANnel<n> for the DSO models

<source> ::= {DIGital0,..,DIGital15 | POD1 | POD2 | CHANnel<n>} for the MSO models

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models

The <source> parameter may be repeated up to 5 times.
```

The :AUToscale command evaluates all input signals and sets the correct conditions to display the signals.

This is the same as pressing the Autoscale key on the front panel.

If one or more sources are specified, those specified sources will be enabled and all others blanked. The autoscale channels mode (see :AUToscale:CHANnels) is set to DISPlayed channels. Then, the autoscale is performed.

When the :AUToscale command is sent, the following conditions are affected and actions are taken:

- Thresholds.
- Channels with activity around the trigger point are turned on, others are turned off.
- Channels are reordered on screen; analog channel 1 first, followed by the remaining analog channels, then the digital channels 0-15.
- Delay is set to 0 seconds.
- Time/Div.

The :AUToscale command does not affect the following conditions:

- Label names.
- Trigger conditioning.

The :AUToscale command turns off the following items:

- Cursors.
- Measurements.
- Trace memories.
- Delayed time base mode.

For further information on :AUToscale, see the *User's Guide*.

See Also

- Introduction to Root (:) Commands
- :AUToscale:CHANnels
- :AUToscale:AMODE

Example Code

```
' AUTOSCALE - This command evaluates all the input signals and sets
' the correct conditions to display all of the active signals.
myScope.WriteString ":AUTOSCALE"   ' Same as pressing the Autoscale key.
```

Example program from the start: VISA COM Example in Visual Basic

Root (:) Commands

:AUToscale:AMODE — N

Command Syntax

```
:AUToscale:AMODE <value>
```

```
<value> ::= {NORMal | CURRent}
```

The :AUTOscale:AMODE command specifies the acquisition mode that is set by subsequent :AUToscales.

- When NORMal is selected, an :AUToscale command sets the NORMal acquisition type and the RTIMe (real-time) acquisition mode.
- When CURRent is selected, the current acquisition type and mode are kept on subsequent :AUToscales.

Use the :ACQuire:TYPE and :ACQuire:MODE commands to set the acquisition type and mode.

Query Syntax

```
:AUToscale:AMODE?
```

The :AUToscale:AMODE? query returns the autoscale acquire mode setting.

Return Format

```
<value><NL>
```

```
<value> ::= {NORM | CURR}
```

See Also

- Introduction to Root (:) Commands
- :AUToscale
- :AUToscale:CHANnels
- :ACQuire:TYPE
- :ACQuire:MODE

Root (:) Commands

:AUToscale:CHANnels — N

Command Syntax

```
:AUToscale:CHANnels <value>
```

```
<value> ::= {ALL | DISPlayed}
```

The :AUTOscale:CHANnels command specifies which channels will be displayed on subsequent :AUToscales.

- When ALL is selected, all channels that meet the requirements of :AUToscale will be displayed.
- When DISPlayed is selected, only the channels that are turned on are autoscaled.

Use the :VIEW or :BLANk root commands to turn channels on or off.

Query Syntax

```
:AUToscale:CHANnels?
```

The :AUToscale:CHANnels? query returns the autoscale channels setting.

Return Format

```
<value><NL>

<value> ::= {ALL | DISP}
```

See Also

- Introduction to Root (:) Commands
- :AUToscale
- :AUToscale:AMODE
- :VIEW
- :BLANk

Root (:) Commands

:BLANk — N

Command Syntax

```
:BLANk [<source>]

<source> ::= {CHANnel<n> | FUNCtion | MATH | SBUS} for the DSO models

<source> ::= {CHANnel<n> | DIGital0,..,DIGital15 | POD{1 | 2} | BUS{1 | 2}
| FUNCtion | MATH | SBUS} for the MSO models

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :BLANk command turns off (stops displaying) the specified channel, digital pod, math function, or serial decode bus. The :BLANk command with no parameter turns off all sources.

NOTE To turn on (start displaying) a channel, etc., use the :VIEW command. The DISPlay commands, :CHANnel<n>:DISPlay, :FUNCtion:DISPlay, :POD<n>:DISPlay, or :DIGital<n>:DISPlay, are the preferred method to turn on/off a channel, etc.

NOTE MATH is an alias for FUNCtion.

See Also

- Introduction to Root (:) Commands
- :CDISplay
- :CHANnel<n>:DISPlay
- :DIGital<n>:DISPlay
- :FUNCtion:DISPlay
- :POD<n>:DISPlay
- :STATus
- :VIEW

Example Code

- Example Code

Root (:) Commands

:CDISplay — C

Command Syntax

```
:CDISplay
```

The :CDISplay command clears the display and resets all associated measurements. If the oscilloscope is stopped, all currently displayed data is erased. If the oscilloscope is running, all the data in active channels and functions is erased; however, new data is displayed on the next acquisition.

See Also

- Introduction to Root (:) Commands
- :DISPlay:CLEar

Root (:) Commands

:DIGitize — C

Command Syntax

```
:DIGitize [<source>[,..,<source>]]

<source> ::= {CHANnel<n> | FUNCtion | MATH | SBUS} for the DSO models

<source> ::= {CHANnel<n> | DIGital0,..,DIGital15 | POD{1 | 2} | BUS{1 | 2}
| FUNCtion | MATH | SBUS} for the MSO models

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

```
The <source> parameter may be repeated up to 5 times.
```

The :DIGitize command is a specialized RUN command. It causes the instrument to acquire waveforms according to the settings of the :ACQuire Commands subsystem. When the acquisition is complete, the instrument is stopped. If no argument is given, DIGitize acquires the channels currently displayed. If no channels are displayed, all channels are acquired.

NOTE To halt a DIGitize in progress, use the device clear command.

NOTE MATH is an alias for FUNCtion.

See Also

- Introduction to Root (:) Commands
- :RUN
- :SINGle
- :STOP
- :ACQuire Commands
- :WAVeform Commands

Example Code

```
  ' DIGITIZE - Used to acquire the waveform data for transfer over
  ' the interface.  Sending this command causes an acquisition to
  ' take place with the resulting data being placed in the buffer.
  '
  ' NOTE!  The DIGITIZE command is highly recommended for triggering
  ' modes other than SINGLE.  This ensures that sufficient data is
  ' available for measurement.  If DIGITIZE is used with single mode,
  ' the completion criteria may never be met.  The number of points
  ' gathered in Single mode is related to the sweep speed, memory
  ' depth, and maximum sample rate.  For example, take an oscilloscope
  ' with a 1000-point memory, a sweep speed of 10 us/div (100 us
  ' total time across the screen), and a 20 MSa/s maximum sample rate.
  ' 1000 divided by 100 us equals 10 MSa/s.  Because this number is
  ' less than or equal to the maximum sample rate, the full 1000 points
  ' will be digitized in a single acquisition.  Now, use 1 us/div
  ' (10 us across the screen).  1000 divided by 10 us equals 100 MSa/s;
  ' because this is greater than the maximum sample rate by 5 times,
  ' only 400 points (or 1/5 the points) can be gathered on a single
  ' trigger.  Keep in mind when the oscilloscope is running, communication
  ' with the computer interrupts data acquisition.  Setting up the
  ' oscilloscope over the bus causes the data buffers to be cleared
  ' and internal hardware to be reconfigured.  If a measurement is
  ' immediately requested, there may have not been enough time for
  ' the data acquisition process to collect data, and the results may
  ' not be accurate.  An error value of 9.9E+37 may be returned over
  ' the bus in this situation.
  '
  myScope.WriteString ":DIGITIZE CHAN1"
```

Example program from the start: VISA COM Example in Visual Basic

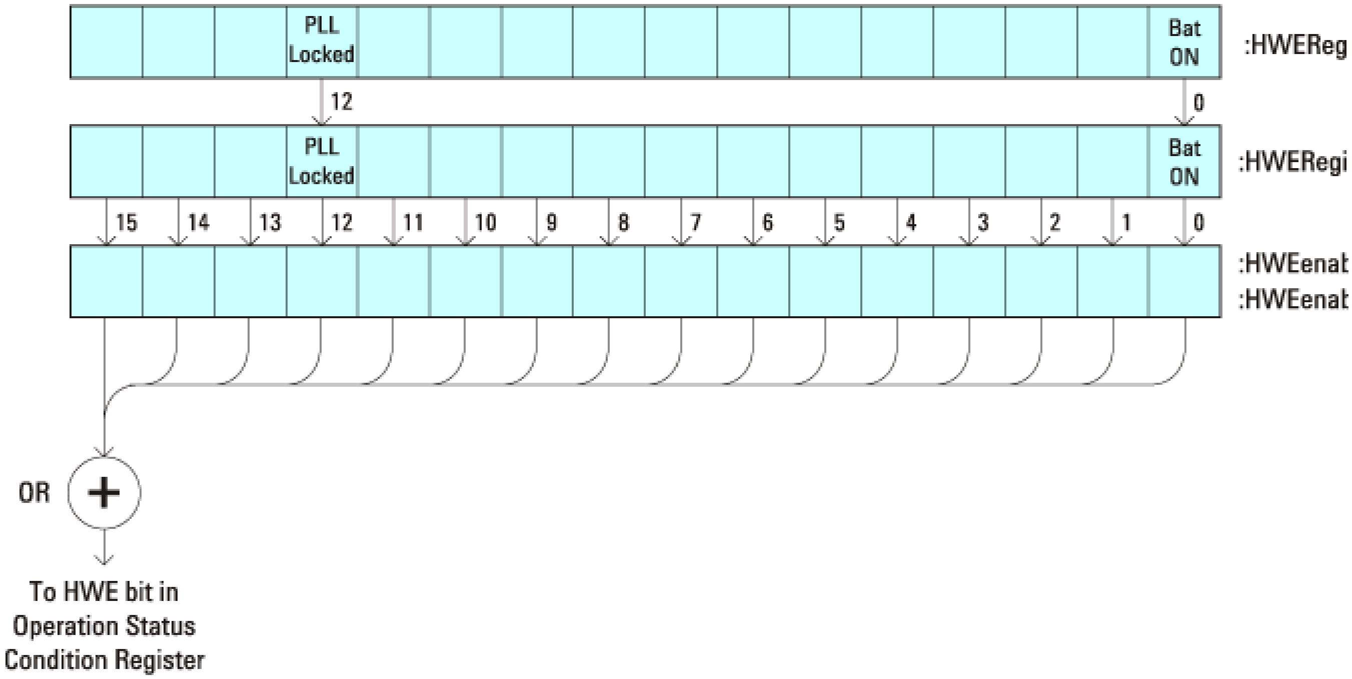
:HWEenable (Hardware Event Enable Register) — N

Command Syntax

```
:HWEenable <mask>
```

```
<mask> ::= 16-bit integer
```

The :HWEenable command sets a mask in the Hardware Event Enable register. Set any of the following bits to "1" to enable bit 12 in the Operation Status Condition Register and potentially cause an SRQ (Service Request interrupt to be generated.



Hardware Event Enable Register (HWEenable):

Bit	Name	Description	When Set (1 = High = True), Enables:
15-13	---	---	(Not used.)
12	PLL Locked	PLL Locked	This bit is for internal use and is not intended for general use.
11-1	---	---	(Not used.)
0	Bat On	Battery On	Event when the battery is on.

Query Syntax

```
:HWEenable?
```

The :HWEenable? query returns the current value contained in the Hardware Event Enable register as an integer number.

Return Format

```
<value><NL>

<value> ::= integer in NR1 format.
```

See Also

- Introduction to Root (:) Commands
- :AER (Arm Event Register)
- :CHANnel<n>:PROTection
- :EXTernal:PROTection
- :OPERegister[:EVENt] (Operation Status Event Register)
- :OVLenable (Overload Event Enable Register)
- :OVLRegister (Overload Event Register)
- *STB (Read Status Byte)
- *SRE (Service Request Enable)

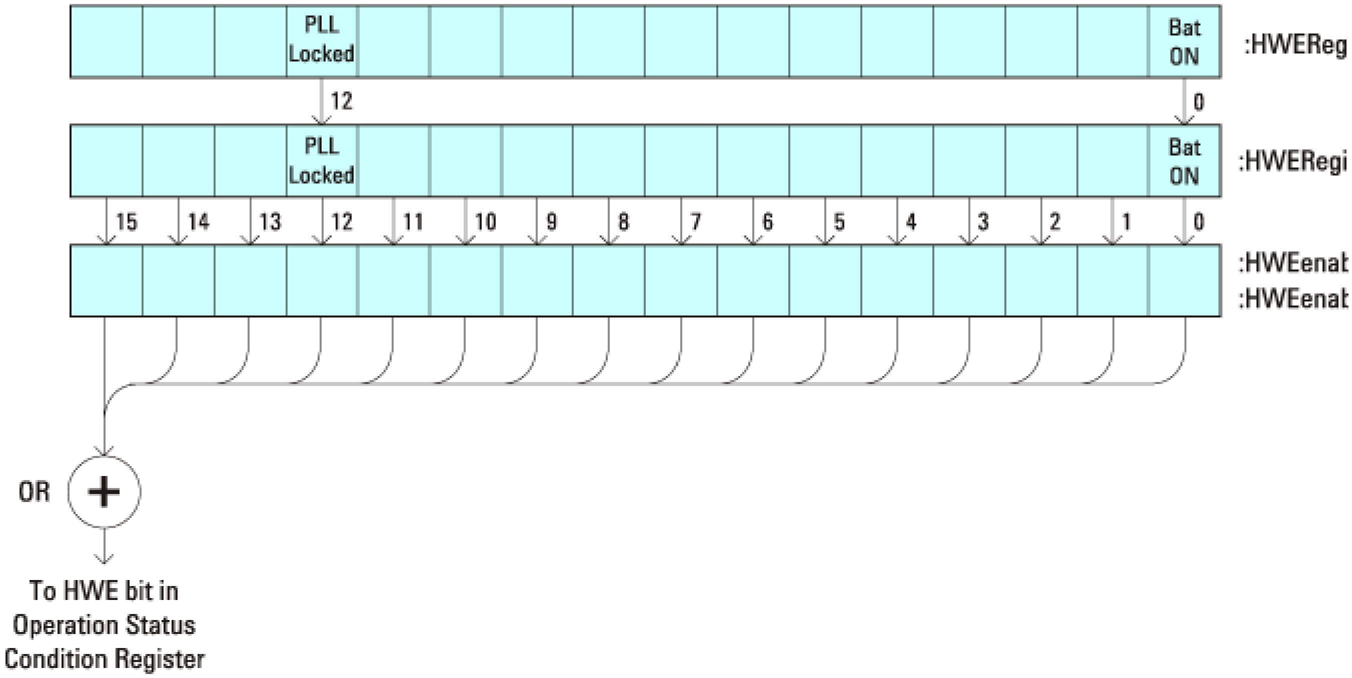
Root (:) Commands

:HWERegister:CONDition (Hardware Event Condition Register) — N

Query Syntax

```
:HWERegister:CONDition?
```

The :HWERegister:CONDition? query returns the integer value contained in the Hardware Event Condition Register.



Hardware Event Condition Register:

Bit	Name	Description	When Set (1 = High = True), Indicates:
15-13	---	---	(Not used.)
12	PLL Locked	PLL Locked	This bit is for internal use and is not intended for general use.
11-1	---	---	(Not used.)
0	Bat On	Battery On	The battery is on.

Return Format

```
<value><NL>

<value> ::= integer in NR1 format.
```

See Also

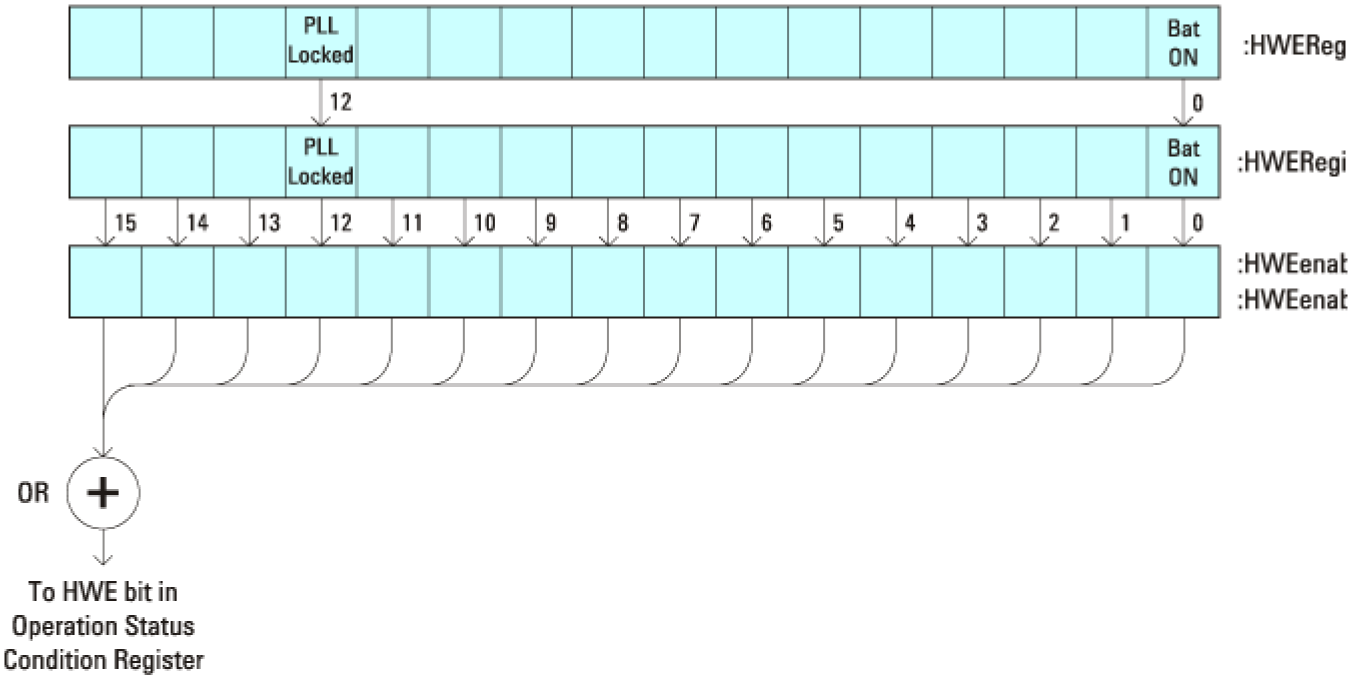
- Introduction to Root (:) Commands
- :CHANnel<n>:PROTection
- :EXTernal:PROTection
- :OPEE (Operation Status Enable Register)
- :OPERegister[:EVENt] (Operation Status Event Register)
- :OVLenable (Overload Event Enable Register)
- :OVLRegister (Overload Event Register)
- *STB (Read Status Byte)
- *SRE (Service Request Enable)

:HWERegister[:EVENt] (Hardware Event Event Register) — N

Query Syntax

```
:HWERegister[:EVENt]?
```

The :HWERegister[:EVENt]? query returns the integer value contained in the Hardware Event Event Register.



Hardware Event Event Register:

Bit	Name	Description	When Set (1 = High = True), Indicates:
15-13	---	---	(Not used.)
12	PLL Locked	PLL Locked	This bit is for internal use and is not intended for general use.
11-1	---	---	(Not used.)
0	Bat On	Battery On	The battery is on.

Return Format

```
<value><NL>

<value> ::= integer in NR1 format.
```

See Also

- Introduction to Root (:) Commands
- :CHANnel<n>:PROTection
- :EXTernal:PROTection
- :OPEE (Operation Status Enable Register)
- :OPERegister:CONDition (Operation Status Condition Register)
- :OVLenable (Overload Event Enable Register)
- :OVLRegister (Overload Event Register)
- *STB (Read Status Byte)
- *SRE (Service Request Enable)

Root (:) Commands

:MERGe — N

Command Syntax

```
:MERGe <pixel memory>

<pixel memory> ::= {PMEMory0 | PMEMory1 | PMEMory2 | PMEMory3 | PMEMory4
| PMEMory5 | PMEMory6 | PMEMory7 | PMEMory8 | PMEMory9}
```

The :MERGe command stores the contents of the active display in the specified pixel memory. The previous contents of the pixel memory are overwritten. The pixel memories are PMEMory0 through PMEMory9. This command is similar to the function of the "Save To: INTERN_<n>" key in the Save/Recall menu.

See Also

- Introduction to Root (:) Commands
- *SAV (Save)
- *RCL (Recall)
- :VIEW
- :BLANk

Root (:) Commands

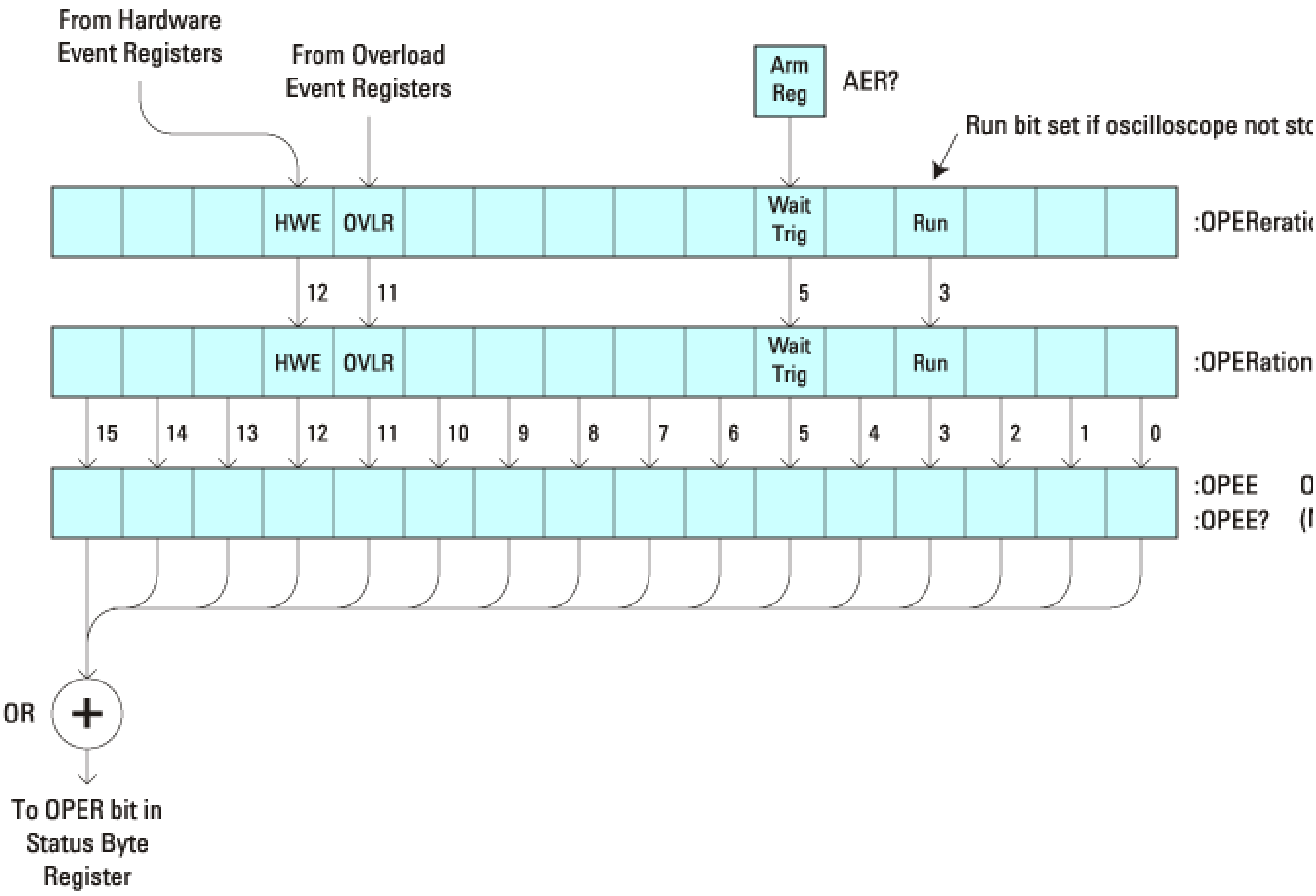
:OPEE (Operation Status Enable Register) — C

Command Syntax

```
:OPEE<mask>

<mask> ::= 16-bit integer
```

The :OPEE command sets a mask in the Operation Status Enable register. Set any of the following bits to "1" to enable bit 7 in the Status Byte Register and potentially cause an SRQ (Service Request interrupt to be generated.


From Hardware Event Registers
From Overload Event Registers
Arm Reg
AER?
Run bit set if oscilloscope not sto
HWE
OVLR
Wait Trig
Run
:OPERati
12
11
5
3
HWE
OVLR
Wait Trig
Run
:OPERation
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
:OPEE 0
:OPEE? (
OR
+
To OPER bit in Status Byte Register

Operation Status Enable Register (OPEE):

Bit	Name	Description	When Set (1 = High = True), Enables:
15-13	---	---	(Not used.)
12	HWE	Hardware Event	Event when hardware event occurs.
11	OVLR	Overload	Event when 50Ω input overload occurs.
10-6	---	---	(Not used.)
5	Wait Trig	Wait Trig	Event when the trigger is armed.
4	---	---	(Not used.)
3	Run	Running	Event when the oscilloscope is running (not stopped).
2-0	---	---	(Not used.)

Query Syntax

```
:OPEE?
```

The :OPEE? query returns the current value contained in the Operation Status Enable register as an integer number.

Return Format

```
<value><NL>

<value> ::= integer in NR1 format.
```

See Also

- Introduction to Root (:) Commands
- :AER (Arm Event Register)
- :CHANnel<n>:PROTection
- :EXTernal:PROTection
- :OPERegister[:EVENt] (Operation Status Event Register)
- :OVLenable (Overload Event Enable Register)
- :OVLRegister (Overload Event Register)
- *STB (Read Status Byte)
- *SRE (Service Request Enable)

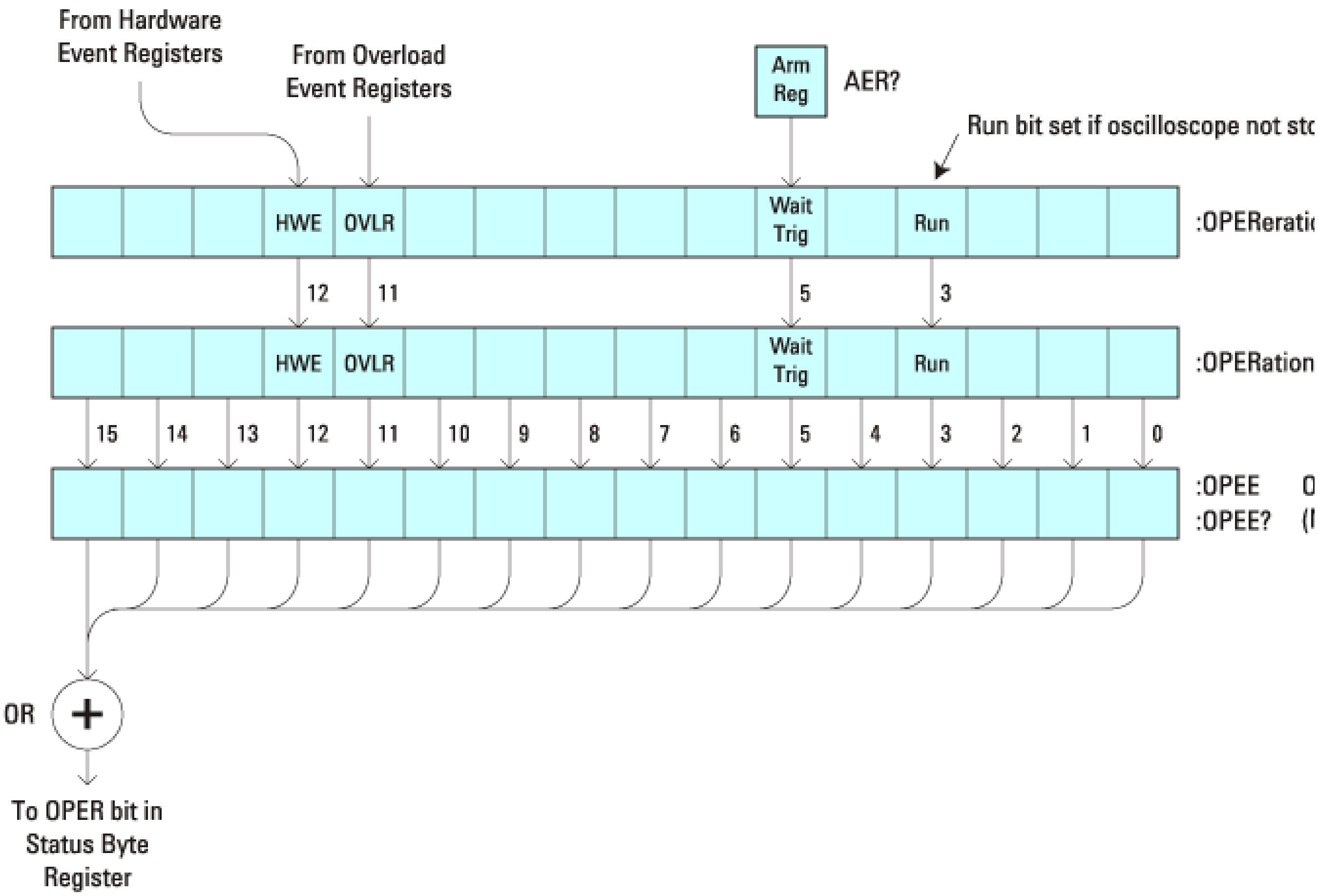
Root (:) Commands

:OPERegister:CONDition (Operation Status Condition Register) — C

Query Syntax

```
:OPERegister:CONDition?
```

The :OPERegister:CONDition? query returns the integer value contained in the Operation Status Condition Register.


From Hardware Event Registers
From Overload Event Registers
Arm Reg
AER?
Run bit set if oscilloscope not stc
HWE
OVLR
Wait Trig
Run
:OPERratic
12
11
5
3
HWE
OVLR
Wait Trig
Run
:OPERation
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
:OPEE
:OPEE?
0
(I
OR
To OPER bit in Status Byte Register

Operation Status Condition Register:

Bit	Name	Description	When Set (1 = High = True), Indicates:
15-13	---	---	(Not used.)
12	HWE	Hardware Event	A hardware event has occurred..
11	OVLR	Overload	A 50Ω input overload has occurred.
10-6	---	---	(Not used.)
5	Wait Trig	Wait Trig	The trigger is armed (set by the Trigger Armed Event Register (TER)).
4	---	---	(Not used.)
3	Run	Running	The oscilloscope is running (not stopped).
2-0	---	---	(Not used.)

Return Format

`<value><NL>`

`<value> ::= integer in NR1 format.`

See Also

- Introduction to Root (:) Commands
- :CHANNel<n>:PROTection
- :EXTernal:PROTection
- :OPEE (Operation Status Enable Register)
- :OPERegister[:EVENt] (Operation Status Event Register)
- :OVLenable (Overload Event Enable Register)
- :OVLRegister (Overload Event Register)
- *STB (Read Status Byte)
- *SRE (Service Request Enable)
- :HWERegister[:EVENt] (Hardware Event Event Register)
- :HWEenable (Hardware Event Enable Register)

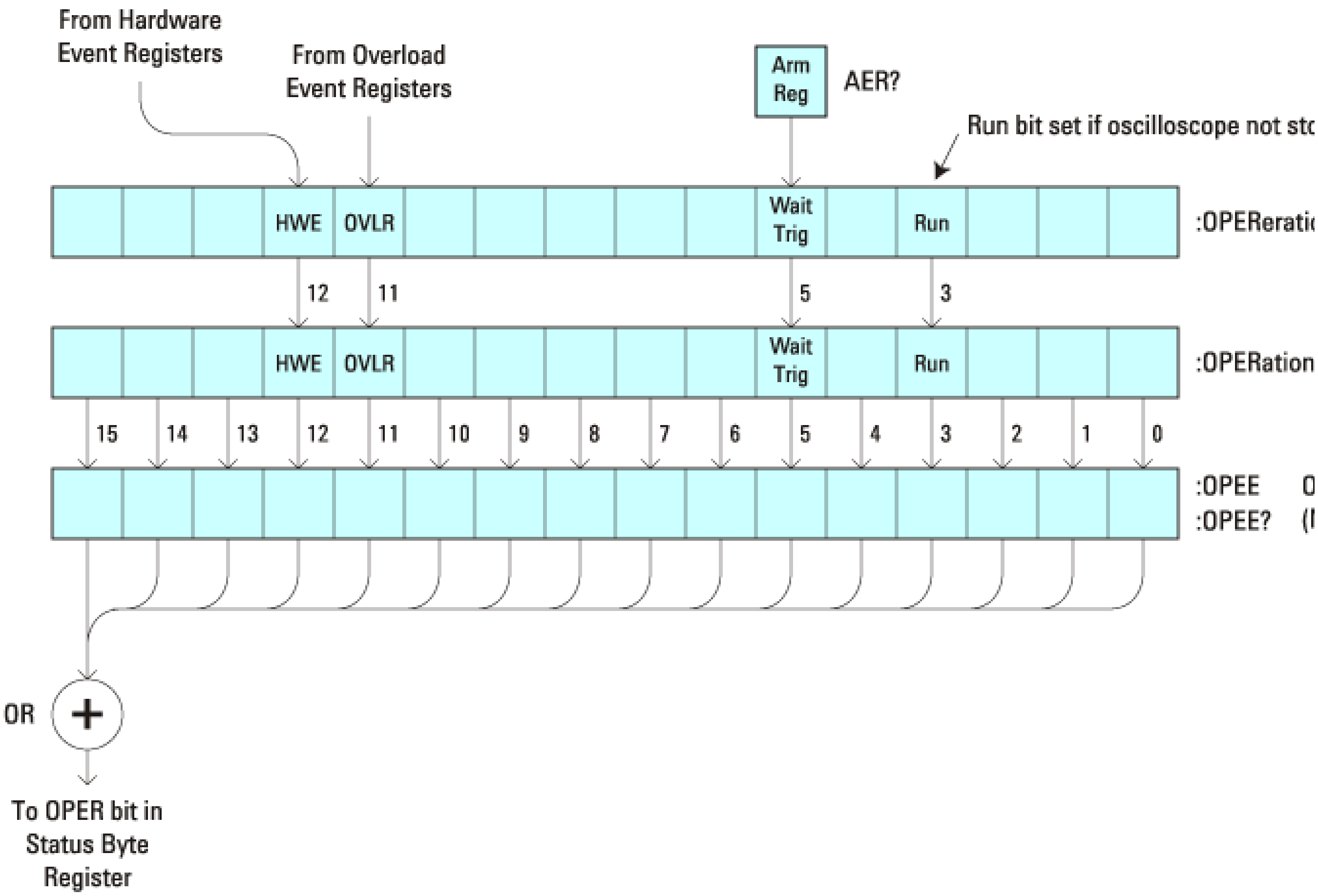
Root (:) Commands

:OPERegister[:EVENt] (Operation Status Event Register) — C

Query Syntax

```
:OPERegister[:EVENt]?
```

The :OPERegister[:EVENt]? query returns the integer value contained in the Operation Status Event Register.


From Hardware Event Registers
From Overload Event Registers
Arm Reg
AER?
Run bit set if oscilloscope not stc
HWE
OVLR
Wait Trig
Run
:OPERerati
12
11
5
3
HWE
OVLR
Wait Trig
Run
:OPERation
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
:OPEE 0
:OPEE? (I
OR
+
To OPER bit in Status Byte Register

Operation Status Event Register:

Bit	Name	Description	When Set (1 = High = True), Indicates:
15-13	---	---	(Not used.)
12	HWE	Hardware Event	A hardware event has occurred.
11	OVLR	Overload	A 50Ω input overload has occurred.
10-6	---	---	(Not used.)
5	Wait Trig	Wait Trig	The trigger is armed (set by the Trigger Armed Event Register (TER)).
4	---	---	(Not used.)
3	Run	Running	The oscilloscope has gone from a stop state to a single or running state.
2-0	---	---	(Not used.)

Return Format

`<value><NL>`

```
<value> ::= integer in NR1 format.
```

See Also

- Introduction to Root (:) Commands
- :CHANnel<n>:PROTection
- :EXTernal:PROTection
- :OPEE (Operation Status Enable Register)
- :OPERegister:CONDition (Operation Status Condition Register)
- :OVLenable (Overload Event Enable Register)
- :OVLRegister (Overload Event Register)
- *STB (Read Status Byte)
- *SRE (Service Request Enable)
- :HWERegister[:EVENt] (Hardware Event Event Register)
- :HWEenable (Hardware Event Enable Register)

Root (:) Commands

:OVLenable (Overload Event Enable Register) — C

Command Syntax

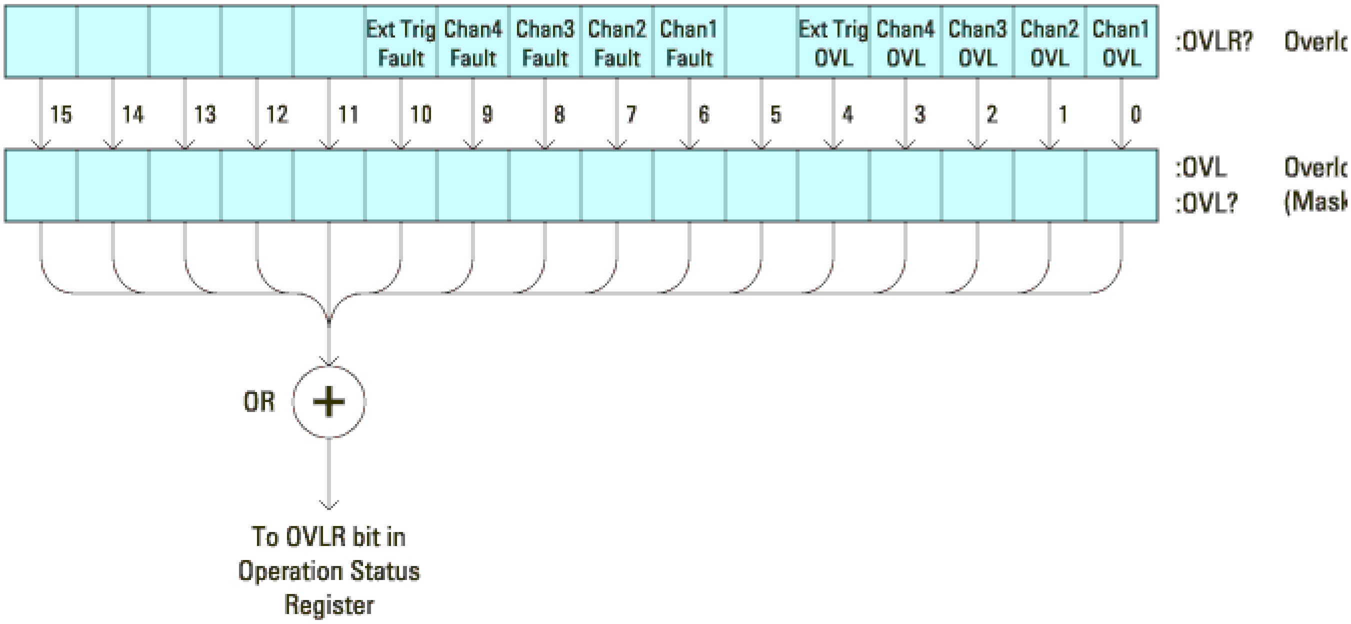
```
:OVLenable <enable_mask>

<enable_mask> ::= 16-bit integer
```

The overload enable mask is an integer representing an input as described in the following table.

The :OVLenable command sets the mask in the Overload Event Enable Register and enables the reporting of the Overload Event Register. If an overvoltage is sensed on a 50Ω input, the input will automatically switch to 1 MΩ input impedance. If enabled, such an event will set bit 11 in the Operation Status Register.

NOTE You can set analog channel input impedance to 50Ω on the 300 MHz, 500 MHz, and 1 GHz bandwidth oscilloscope models. On these same bandwidth models, if there are only two analog channels, you can also set external trigger input impedance to 50Ω.



Overload Event Enable Register (OVL):

Bit	Description	When Set (1 = High = True), Enables:
15-11	---	(Not used.)
10	External Trigger Fault	Event when fault occurs on External Trigger input.
9	Channel 4 Fault	Event when fault occurs on Channel 4 input.
8	Channel 3 Fault	Event when fault occurs on Channel 3 input.
7	Channel 2 Fault	Event when fault occurs on Channel 2 input.
6	Channel 1 Fault	Event when fault occurs on Channel 1 input.
5	---	(Not used.)
4	External Trigger OVL	Event when overload occurs on External Trigger input.
3	Channel 4 OVL	Event when overload occurs on Channel 4 input.
2	Channel 3 OVL	Event when overload occurs on Channel 3 input.
1	Channel 2 OVL	Event when overload occurs on Channel 2 input.
0	Channel 1 OVL	Event when overload occurs on Channel 1 input.

Query Syntax

```
:OVLenable?
```

The :OVLenable query returns the current enable mask value contained in the Overload Event Enable Register.

Return Format

```
<enable_mask><NL>
```

```
<enable_mask> ::= integer in NR1 format.
```

See Also

- Introduction to Root (:) Commands
- :CHANNel<n>:PROTection
- :EXTernal:PROTection
- :OPEE (Operation Status Enable Register)
- :OPERegister:CONDition (Operation Status Condition Register)
- :OPERegister[:EVENt] (Operation Status Event Register)
- :OVLRegister (Overload Event Register)
- *STB (Read Status Byte)
- *SRE (Service Request Enable)

Root (:) Commands

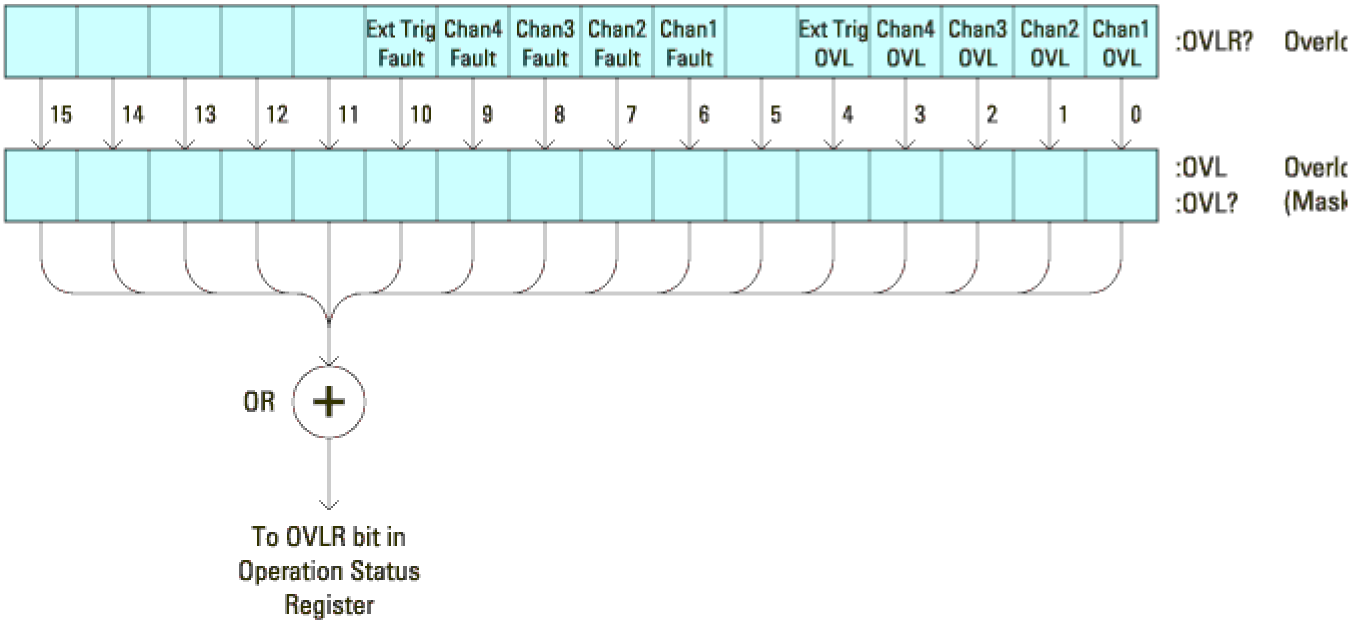
:OVLRegister (Overload Event Register) — C

Query Syntax

```
:OVLRegister?
```

The :OVLRegister query returns the overload protection value stored in the Overload Event Register (OVLR). If an overvoltage is sensed on a 50Ω input, the input will automatically switch to 1 MΩ input impedance. A "1" indicates an overload has occurred.

NOTE You can set analog channel input impedance to 50Ω on the 300 MHz, 500 MHz, and 1 GHz bandwidth oscilloscope models. On these same bandwidth models, if there are only two analog channels, you can also set external trigger input impedance to 50Ω.



Overload Event Register (OVLR):

Bit	Description	When Set (1 = High = True), Indicates:
15-11	---	(Not used.)
10	External Trigger Fault	Fault has occurred on External Trigger input.
9	Channel 4 Fault	Fault has occurred on Channel 4 input.
8	Channel 3 Fault	Fault has occurred on Channel 3 input.
7	Channel 2 Fault	Fault has occurred on Channel 2 input.
6	Channel 1 Fault	Fault has occurred on Channel 1 input.
5	---	(Not used.)
4	External Trigger OVL	Overload has occurred on External Trigger input.
3	Channel 4 OVL	Overload has occurred on Channel 4 input.
2	Channel 3 OVL	Overload has occurred on Channel 3 input.
1	Channel 2 OVL	Overload has occurred on Channel 2 input.
0	Channel 1 OVL	Overload has occurred on Channel 1 input.

Return Format

```
<value><NL>

<value> ::= integer in NR1 format.
```

See Also

- Introduction to Root (:) Commands

- :CHANnel<n>:PROTection
- :EXTernal:PROTection
- :OPEE (Operation Status Enable Register)
- :OVLenable (Overload Event Enable Register)
- *STB (Read Status Byte)
- *SRE (Service Request Enable)

Root (:) Commands

:PRINt — C

Command Syntax

```
:PRINt [<options>]

<options> ::= [<print option>][,..,<print option>]

<print option> ::= {COLor | GRAYscale | PRINter0 | BMP8bit | BMP | PNG
| NOFactors | FACTors}

The <print option> parameter may be repeated up to 5 times.
```

The PRINt command formats the output according to the currently selected format (device). If an option is not specified, the value selected in the Print Config menu is used. Refer to :HARDcopy:FORMat for more information.

See Also

- Introduction to Root (:) Commands
- Introduction to :HARDcopy Commands
- :HARDcopy:FORMat
- :HARDcopy:FACTors
- :HARDcopy:GRAYscale
- :DISPlay:DATA

Root (:) Commands

:RUN — C

Command Syntax

```
:RUN
```

The :RUN command starts repetitive acquisitions. This is the same as pressing the Run key on the front panel.

See Also

- Introduction to Root (:) Commands
- :SINGle
- :STOP

Example Code

```
   ' RUN_STOP - (not executed in this example)
   '  - RUN starts the acquisition of data for the active waveform display.
   '  - STOP stops the data acquisition and turns off AUTOSTORE.
   ' myScope.WriteString ":RUN"    ' Start data acquisition.
   ' myScope.WriteString ":STOP"    ' Stop the data acquisition.
```

Example program from the start: VISA COM Example in Visual Basic

Root (:) Commands

:SERial — N

Query Syntax

```
:SERial?
```

The :SERial? query returns the serial number of the instrument.

Return Format:

```
Unquoted string<NL>
```

See Also

- Introduction to Root (:) Commands

Root (:) Commands

:SINGle — C

Command Syntax

```
:SINGle
```

The :SINGle command causes the instrument to acquire a single trigger of data. This is the same as pressing the Single key on the front panel.

See Also

- Introduction to Root (:) Commands
- :RUN
- :STOP

Root (:) Commands

:STATus — N

Query Syntax

```
:STATus? <source>

<source> ::= {CHANnel<n> | FUNCtion | MATH | SBUS} for the DSO models

<source> ::= {CHANnel<n> | DIGital0,..,DIGital15 | POD{1 | 2} | BUS{1 | 2}
| FUNCtion | MATH | SBUS} for the MSO models

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :STATus? query reports whether the channel, function, trace memory, or serial decode bus specified by <source> is displayed.

NOTE MATH is an alias for FUNCtion.

Return Format

```
<value><NL>

<value> ::= {1 | 0}
```

See Also

- Introduction to Root (:) Commands
- :BLANk
- :CHANnel<n>:DISPlay
- :DIGital<n>:DISPlay
- :FUNCtion:DISPlay
- :POD<n>:DISPlay
- :VIEW

Root (:) Commands

:STOP — C

Command Syntax

```
:STOP
```

The :STOP command stops the acquisition. This is the same as pressing the Stop key on the front panel.

See Also

- Introduction to Root (:) Commands
- :RUN
- :SINGle

Example Code

- Example Code

Root (:) Commands

:TER (Trigger Event Register) — C

Query Syntax

```
:TER?
```

The :TER? query reads the Trigger Event Register. After the Trigger Event Register is read, it is cleared. A one indicates a trigger has occurred. A zero indicates a trigger has not occurred.

The Trigger Event Register is summarized in the TRG bit of the Status Byte Register (STB). A Service Request (SRQ) can be generated when the TRG bit of the Status Byte transitions, and the TRG bit is set in the Service Request Enable register. The Trigger Event Register must be cleared each time you want a new service request to be generated.

Return Format

```
<value><NL>

<value> ::= {1 | 0}; a 16-bit integer in NR1 format.
```

See Also

- Introduction to Root (:) Commands
- *SRE (Service Request Enable)
- *STB (Read Status Byte)

Root (:) Commands

:VIEW — N

Command Syntax

```
:VIEW <source>

<source> ::= {CHANnel<n> | PMEMory0,..,PMEMory9 | FUNCtion | MATH
| SBUS} for DSO models
```

```
<source> ::= {CHANnel<n> | DIGital0,..,DIGital15 | PMEMory0,..,PMEMory9
| POD{1 | 2} | BUS{1 | 2} | FUNCtion | MATH | SBUS} for MSO models

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :VIEW command turns on the specified channel, function, trace memory, or serial decode bus.

NOTE MATH is an alias for FUNCtion.

See Also

- Introduction to Root (:) Commands
- :BLANk
- :CHANnel<n>:DISPlay
- :DIGital<n>:DISPlay
- :FUNCtion:DISPlay
- :POD<n>:DISPlay
- :STATus

Example Code

```
  ' VIEW_BLANK - (not executed in this example)
  '  - VIEW turns on (starts displaying) a channel or pixel memory.
  '  - BLANK turns off (stops displaying) a channel or pixel memory.
  ' myScope.WriteString ":BLANK CHANNEL1"    ' Turn channel 1 off.
  ' myScope.WriteString ":VIEW CHANNEL1"    ' Turn channel 1 on.
```

Example program from the start: VISA COM Example in Visual Basic

Commands by Subsystem

:SBUS Commands

Control oscilloscope functions associated with the serial decode bus. See Introduction to :SBUS Commands.

Command	Query	Options and Query Returns
n/a	:SBUS:CAN:COUNt:ERRor?	<frame_count> ::= integer in NR1 format
n/a	:SBUS:CAN:COUNt:OVERload?	<frame_count> ::= integer in NR1 format
:SBUS:CAN:COUNt:RESet	n/a	n/a
n/a	:SBUS:CAN:COUNt:TOTal?	<frame_count> ::= integer in NR1 format
n/a	:SBUS:CAN:COUNt:UTILization?	<percent> ::= floating-point in NR3 format

:SBUS:DISPlay {{0 | OFF} | {1 | ON}}	:SBUS:DISPlay?	{0 | 1}
:SBUS:IIC:ASIZe <size>	:SBUS:IIC:ASIZe?	<size> ::= {BIT7 | BIT8}
:SBUS:LIN:PARity {{0 | OFF} | {1 | ON}}	:SBUS:LIN:PARity?	{0 | 1}
:SBUS:MODE <mode>	:SBUS:MODE?	<mode> ::= {IIC | SPI | CAN | LIN}
:SBUS:SPI:WIDTh <word_width>	:SBUS:SPI:WIDTh?	<word_width> ::= integer 4-16 in NR1 format

Introduction to :SBUS Commands

The :SBUS subsystem commands control the serial decode bus viewing, mode, and other options.

NOTE These commands are only valid on 4-channel or 4+16-channel oscilloscope models when a serial decode option has been licensed.

Reporting the Setup. Use :SBUS? to query setup information for the :SBUS subsystem.

Return Format. The following is a sample response from the :SBUS? query. In this case, the query was issued following a *RST command.

```
:SBUS:DISP 0;MODE IIC
```

:SBUS Commands

:SBUS:CAN:COUNt:ERRor — N

Query Syntax

```
:SBUS:CAN:COUNt:ERRor?
```

Returns the error frame count.

Return Format

```
<frame_count><NL>

<frame_count> ::= integer in NR1 format
```

Errors

- Hardware missing

See Also

- :SBUS:CAN:COUNt:RESet
- Introduction to :SBUS Commands

- :SBUS:MODE
- :TRIGger:CAN Commands

:SBUS Commands

:SBUS:CAN:COUNt:OVERload — N

Query Syntax

```
:SBUS:CAN:COUNt:OVERload?
```

Returns the overload frame count.

Return Format

```
<frame_count><NL>
```

```
<frame_count> ::= integer in NR1 format
```

Errors

- Hardware missing

See Also

- :SBUS:CAN:COUNt:RESet
- Introduction to :SBUS Commands
- :SBUS:MODE
- :TRIGger:CAN Commands

:SBUS Commands

:SBUS:CAN:COUNt:RESet — N

Command Syntax

```
:SBUS:CAN:COUNt:RESet
```

Resets the frame counters.

Errors

- Hardware missing

See Also

- :SBUS:CAN:COUNt:ERRor

- :SBUS:CAN:COUNt:OVERload
- :SBUS:CAN:COUNt:TOTal
- :SBUS:CAN:COUNt:UTILization
- Introduction to :SBUS Commands
- :SBUS:MODE
- :TRIGger:CAN Commands

:SBUS Commands

:SBUS:CAN:COUNt:TOTal — N

Query Syntax

```
:SBUS:CAN:COUNt:TOTal?
```

Returns the total frame count.

Return Format

```
<frame_count><NL>

<frame_count> ::= integer in NR1 format
```

Errors

- Hardware missing

See Also

- :SBUS:CAN:COUNt:RESet
- Introduction to :SBUS Commands
- :SBUS:MODE
- :TRIGger:CAN Commands

:SBUS Commands

:SBUS:CAN:COUNt:UTILization — N

Query Syntax

```
:SBUS:CAN:COUNt:UTILization?
```

Returns the percent utilization.

Return Format

```
<percent><NL>
```

```
<percent> ::= floating-point in NR3 format
```

Errors

- Hardware missing

See Also

- :SBUS:CAN:COUNt:RESet
- Introduction to :SBUS Commands
- :SBUS:MODE
- :TRIGger:CAN Commands

:SBUS Commands

:SBUS:DISPlay — N

Command Syntax

```
:SBUS:DISPlay <display>
```

```
<display> ::= {{1 | ON} | {0 | OFF}}
```

The :SBUS:DISPlay command turns displaying of the serial decode bus on or off.

NOTE This command is only valid on 4-channel or 4+16-channel oscilloscope models when a serial decode option has been licensed.

Query Syntax

```
:SBUS:DISPlay?
```

The :SBUS:DISPlay? query returns the current display setting of the serial decode bus.

Return Format

```
<display><NL>
```

```
<display> ::= {0 | 1}
```

Errors

- Hardware missing

See Also

- Introduction to :SBUS Commands

- :CHANnel<n>:DISPlay
- :DIGital<n>:DISPlay
- :POD<n>:DISPlay
- :VIEW
- :BLANk
- :STATus

:SBUS Commands

:SBUS:IIC:ASIZe — N

Command Syntax

```
:SBUS:IIC:ASIZe <size>

<size> ::= {BIT7 | BIT8}
```

The :SBUS:IIC:ASIZe command determines whether the Read/Write bit is included as the LSB in the display of the IIC address field of the decode bus.

NOTE This command is only valid on 4-channel or 4+16-channel oscilloscope models when the low-speed IIC and SPI serial decode option (Option LSS) has been licensed.

Query Syntax

```
:SBUS:IIC:ASIZe?
```

The :SBUS:IIC:ASIZe? query returns the current IIC address width setting.

Return Format

```
<mode><NL>

<mode> ::= {BIT7 | BIT8}
```

Errors

- Hardware missing

See Also

- Introduction to :SBUS Commands
- :TRIGger:IIC Commands

:SBUS Commands

:SBUS:LIN:PARity — N

Command Syntax

```
:SBUS:LIN:PARity <display>

<display> ::= {{1 | ON} | {0 | OFF}}
```

The :SBUS:LIN:PARity command determines whether the parity bits are included as the most significant bits (MSB) in the display of the Frame Id field in the LIN decode bus.

NOTE This command is only valid on 4-channel or 4+16-channel oscilloscope models when the automotive CAN and LIN serial decode option (Option AMS) has been licensed.

Query Syntax

```
:SBUS:LIN:PARity?
```

The :SBUS:LIN:PARity? query returns the current LIN parity bits display setting of the serial decode bus.

Return Format

```
<display><NL>

<display> ::= {0 | 1}
```

Errors

- Hardware missing

See Also

- Introduction to :SBUS Commands
- :TRIGger:LIN Commands

:SBUS Commands

:SBUS:MODE — N

Command Syntax

```
:SBUS:MODE <mode>

<mode> ::= {IIC | SPI | CAN | LIN}
```

The :SBUS:MODE command determines the decode mode for the serial bus.

This command is only valid on 4-channel or 4+16-channel oscilloscope models when a serial decode option has been licensed.

NOTE

Query Syntax

```
:SBUS:MODE?
```

The :SBUS:MODE? query returns the current serial bus decode mode setting.

Return Format

```
<mode><NL>

<mode> ::= {IIC | SPI | CAN | LIN | NONE}
```

Errors

- Hardware missing

See Also

- Introduction to :SBUS Commands
- :TRIGger:MODE
- :TRIGger:IIC Commands
- :TRIGger:SPI Commands
- :TRIGger:CAN Commands
- :TRIGger:LIN Commands

:SBUS Commands

:SBUS:SPI:WIDTh — N

Command Syntax

```
:SBUS:SPI:WIDTh <word_width>

<word_width> ::= integer 4-16 in NR1 format
```

The :SBUS:SPI:WIDTh command determines the number of bits in a word of data for SPI.

NOTE This command is only valid on 4-channel or 4+16-channel oscilloscope models when the low-speed IIC and SPI serial decode option (Option LSS) has been licensed.

Query Syntax

```
:SBUS:SPI:WIDTh?
```

The :SBUS:SPI:WIDTh? query returns the current SPI decode word width.

Return Format

`<word_width><NL>`

`<word_width> ::= integer 4-16 in NR1 format`

Errors

- Hardware missing

See Also

- Introduction to :SBUS Commands
- :SBUS:MODE
- :TRIGger:SPI Commands

Commands by Subsystem

:SYSTem Commands

Control basic system functions of the oscilloscope. See Introduction to :SYSTem Commands.

Command	Query	Options and Query Returns

:SYSTem:DATE <date>	:SYSTem:DATE?	<date> ::= <year>,<month>,<day>
		<year> ::= 4-digit year in NR1 format
		<month> ::= {1,..,12 | JANuary | FEBruary | MARch | APRil | MAY | JUNe | JULy | AUGust | SEPtember | OCTober | NOVember | DECember}
		<day> ::= {1,..31}

:SYSTem:DSP <string>	n/a	<string> ::= up to 254 characters as a quoted ASCII string
n/a	:SYSTem:ERRor?	<error> ::= an integer error code <error string> ::= quoted ASCII string. See Error Messages.
:SYSTem:LOCK	:SYSTem:LOCK?	<value> ::= {ON | OFF}
:SYSTem:SETup <setup_data>	:SYSTem:SETup?	<setup_data> ::= data in IEEE 488.2 # format.
:SYSTem:TIME <time>	:SYSTem:TIME?	<time> ::= hours,minutes,seconds in NR1 format

Introduction to :SYSTem Commands

SYSTem subsystem commands enable writing messages to the display, setting and reading both the time and the date, querying for errors, and saving and recalling setups.

:SYSTem Commands

:SYSTem:DATE — N

Command Syntax

```
:SYSTem:DATE <date>

<date> ::= <year>,<month>,<day>

<year> ::= 4-digit year in NR1 format

<month> ::= {1,..,12 | JANuary | FEBruary | MARch | APRil | MAY | JUNe | JULy |
AUGust | SEPtember | OCTober | NOVember | DECember}

<day> ::= {1,..,31}
```

The :SYSTem:DATE command sets the date. Validity checking is performed to ensure that the date is valid.

Query Syntax

```
:SYSTem:DATE?
```

The SYSTem:DATE? query returns the date.

Return Format

```
<year>,<month>,<day><NL>
```

See Also

- Introduction to :SYSTem Commands
- :SYSTem:TIME

:SYSTem Commands

:SYSTem:DSP — N

Command Syntax

```
:SYSTem:DSP <string>

<string> ::= quoted ASCII string (up to 254 characters)
```

The :SYSTem:DSP command writes the quoted string (excluding quotation marks) to a text box in the center of the display. Use :SYStem:DSP "" to remotely remove the message from the display. (Two sets of quote marks without a space between them creates a NULL string.) Press any menu key to manually remove the message from the display.

See Also

- Introduction to :SYSTem Commands

:SYSTem Commands

:SYSTem:ERRor — C

Query Syntax

```
:SYSTem:ERRor?
```

The :SYSTem:ERRor? query outputs the next error number and text from the error queue. The instrument has an error queue that is 30 errors deep and operates on a first-in, first-out basis. Repeatedly sending the :SYSTem:ERRor? query returns the errors in the order that they occurred until the queue is empty. Any further queries then return zero until another error occurs.

Return Format

```
<error number>,<error string><NL>

<error number> ::= an integer error code in NR1 format

<error string> ::= quoted ASCII string containing the error message
```

Error messages are listed in *Error Messages*.

See Also

- Introduction to :SYSTem Commands
- *ESR (Standard Event Status Register)
- *CLS (Clear Status)

:SYSTem Commands

:SYSTem:LOCK — N

Command Syntax

```
:SYSTem:LOCK <value>

<value> ::= {{1 | ON} | {0 | OFF}}
```

The :SYSTem:LOCK command disables the front panel. LOCK ON is the equivalent of sending a local lockout message over GPIB.

Query Syntax

```
:SYSTem:LOCK?
```

The :SYSTem:LOCK? query returns the lock status of the front panel.

Return Format

```
<value><NL>

<value> ::= {1 | 0}
```

See Also

- Introduction to :SYSTem Commands

:SYSTem Commands

:SYSTem:SETup — C

Command Syntax

```
:SYSTem:SETup <setup_data>

<setup_data> ::= binary block data in IEEE 488.2 # format.
```

The :SYSTem:SETup command sets the oscilloscope as defined by the data in the setup (learn) string sent from the controller. The setup string does not change the interface mode or interface address.

Query Syntax

```
:SYSTem:SETup?
```

The :SYSTem:SETup? query operates the same as the *LRN? query. It outputs the current oscilloscope setup in the form of a learn string to the controller. The setup (learn) string is sent and received as a binary block of data. The format for the data transmission is the # format defined in the IEEE 488.2 specification.

Return Format

```
<setup_data><NL>

<setup_data> ::= binary block data data in IEEE 488.2 # format
```

See Also

- Introduction to :SYSTem Commands
- *LRN (Learn Device Setup)

Example Code

```
  ' SAVE_SYSTEM_SETUP - The :SYSTEM:SETUP? query returns a program message
  ' that contains the current state of the instrument.  Its format is a
  ' definite-length binary block, for example, #800002204<setup string><NL>
  ' where the setup string is 2204 bytes in length.
  myScope.WriteString ":SYSTEM:SETUP?"
  varQueryResult = myScope.ReadIEEEBlock(BinaryType_UI1)
```

```
   CheckForInstrumentErrors   ' After reading query results.
   ' Output setup string to a file:
   Dim strPath As String
   strPath = "c:\scope\config\setup.dat"
   Close #1   ' If #1 is open, close it.
   Open strPath For Binary Access Write Lock Write As #1   ' Open file for output.
   Put #1, , varQueryResult   ' Write data.
   Close #1   ' Close file.

   ' RESTORE_SYSTEM_SETUP - Read the setup string from a file and write it
   ' back to the oscilloscope.
   Dim varSetupString As Variant
   strPath = "c:\scope\config\setup.dat"
   Open strPath For Binary Access Read As #1   ' Open file for input.
   Get #1, , varSetupString   ' Read data.
   Close #1   ' Close file.
   ' Write setup string back to oscilloscope using ":SYSTEM:SETUP" command:
   myScope.WriteIEEEBlock ":SYSTEM:SETUP ", varSetupString
   CheckForInstrumentErrors
```

Example program from the start: VISA COM Example in Visual Basic

:SYSTem Commands

:SYSTem:TIME — N

Command Syntax

```
:SYSTem:TIME <time>

<time> ::= hours,minutes,seconds in NR1 format
```

The :SYSTem:TIME command sets the system time, using a 24-hour format. Commas are used as separators. Validity checking is performed to ensure that the time is valid.

Query Syntax

```
:SYSTem:TIME? <time>
```

The :SYSTem:TIME? query returns the current system time.

Return Format

```
<time><NL>

<time> ::= hours,minutes,seconds in NR1 format
```

See Also

- Introduction to :SYSTem Commands
- :SYSTem:DATE

Commands by Subsystem

:TIMebase Commands

Control all horizontal sweep functions. See Introduction to :TIMebase Commands.

Command	Query	Options and Query Returns
:TIMebase:MODE <value>	:TIMebase:MODE?	<value> ::= {MAIN | WINDow | XY | ROLL}
:TIMebase:POSition <pos>	:TIMebase:POSition?	<pos> ::= time from the trigger event to the display reference point in NR3 format
:TIMebase:RANGe <range_value>	:TIMebase:RANGe?	<range_value> ::= 5 ns through 500 s in NR3 format
:TIMebase:REFClock {{0 | OFF} | {1 | ON}}	:TIMebase:REFClock?	{0 | 1}
:TIMebase:REFerence {LEFT | CENTer | RIGHt}	:TIMebase:REFerence?	<return_value> ::= {LEFT | CENTer | RIGHt}
:TIMebase:SCALe <scale_value>	:TIMebase:SCALe?	<scale_value> ::= scale value in seconds in NR3 format
:TIMebase:VERNier {{0 | OFF} | {1 | ON}}	:TIMebase:VERNier?	{0 | 1}
:TIMebase:WINDow:POSition <pos>	:TIMebase:WINDow:POSition?	<pos> ::= time from the trigger event to the delayed view reference point in NR3 format
:TIMebase:WINDow:RANGe <range_value>	:TIMebase:WINDow:RANGe?	<range value> ::= range value in seconds in NR3 format for the delayed window
:TIMebase:WINDow:SCALe <scale_value>	:TIMebase:WINDow:SCALe?	<scale_value> ::= scale value in seconds in NR3 format for the delayed window

Introduction to :TIMebase Commands

The TIMebase subsystem commands control the horizontal (X-axis) functions and set the oscilloscope to X-Y mode (where channel 1 becomes the X input and channel 2 becomes the Y input). The time per division, delay, vernier control, and reference can be controlled for the main and window (delayed) time bases.

Reporting the Setup. Use :TIMebase? to query setup information for the TIMebase subsystem.

Return Format. The following is a sample response from the :TIMebase? query. In this case, the query was issued following a *RST command.

```
:TIM:MODE MAIN;REF CENT;MAIN:RANG +1.00E-03;POS +0.0E+00
```

:TIMebase Commands

:TIMebase:MODE — C

Command Syntax

```
:TIMebase:MODE <value>

<value> ::= {MAIN | WINDow | XY | ROLL}
```

The :TIMebase:MODE command sets the current time base. There are four time base modes:

- MAIN — The normal time base mode is the main time base. It is the default time base mode after the *RST (Reset) command.
- WINDow — In the WINDow (delayed) time base mode, measurements are made in the delayed time base if possible; otherwise, the measurements are made in the main time base.
- XY — In the XY mode, the :TIMebase:RANGe, :TIMebase:POSition, and :TIMebase:REFerence commands are not available. No measurements are available in this mode.
- ROLL — In the ROLL mode, data moves continuously across the display from left to right. The oscilloscope runs continuously and is untriggered. The :TIMebase:REFerence selection changes to RIGHt.

NOTE If a :DIGitize command is executed when the :TIMebase:MODE is not MAIN, the :TIMebase:MODE is set to MAIN.

Query Syntax

```
:TIMebase:MODE?
```

The :TIMebase:MODE query returns the current time base mode.

Return Format

```
<value><NL>

<value> ::= {MAIN | WIND | XY | ROLL}
```

See Also

- Introduction to :TIMebase Commands
- *RST (Reset)

Example Code

```
    ' TIMEBASE_MODE - (not executed in this example)
    ' Set the time base mode to MAIN, DELAYED, XY, or ROLL.
    ' myScope.WriteString ":TIMEBASE:MODE MAIN"   ' Set time base mode to main.
```

Example program from the start: VISA COM Example in Visual Basic

:TIMebase Commands

:TIMebase:POSition — C

Command Syntax

```
:TIMebase:POSition <pos>
```

```
<pos> ::= time from the trigger to the display reference in seconds
```

The :TIMebase:POSition command sets the time interval between the trigger event and the display reference point on the screen. The display reference point is either left, right, or center and is set with the :TIMebase:REFerence command. The maximum position value depends on the time/division settings.

NOTE This command is an alias for the :TIMebase:DELay command.

Query Syntax

```
:TIMebase:POSition?
```

The :TIMebase:POSition? query returns the current time from the trigger to the display reference in seconds.

Return Format

```
<pos><NL>
```

```
<pos> ::= time from the trigger to the display reference in seconds
```

See Also

- Introduction to :TIMebase Commands
- :TIMebase:REFerence
- :TIMebase:RANGe
- :TIMebase:SCALe
- :TIMebase:WINDow:POSition

:TIMebase Commands

:TIMebase:RANGe — C

Command Syntax

```
:TIMebase:RANGe <range_value>
```

```
<range_value> ::= 5 ns through 500 s in NR3 format
```

The :TIMebase:RANGe command sets the full-scale horizontal time in seconds for the main window. The range is 10 times the current time-per-division setting.

Query Syntax

```
:TIMebase:RANGe?
```

The :TIMebase:RANGe query returns the current full-scale range value for the main window.

Return Format

```
<range_value><NL>

<range_value> ::= 5 ns through 500 s in NR3 format
```

See Also

- Introduction to :TIMebase Commands
- :TIMebase:MODE
- :TIMebase:SCALe
- :TIMebase:WINDow:RANGe

Example Code

```
   ' TIME_RANGE - Sets the full scale horizontal time in seconds.  The
   ' range value is 10 times the time per division.
   myScope.WriteString ":TIM:RANG 2e-3"   ' Set the time range to 0.002 seconds.
```

Example program from the start: VISA COM Example in Visual Basic

:TIMebase Commands

:TIMebase:REFClock — N

Command Syntax

```
:TIMebase:REFClock <value>

<value> ::= {{1 | ON} | {0 | OFF}
```

The :TIMebase:REFClock command enables or disables the 10 MHz REF BNC located on the rear panel of the oscilloscope.

The 10 MHz REF BNC can be used as an input for the oscilloscope's reference clock (instead of the internal 10 MHz reference), or it can be used to output the internal 10 MHz reference clock when synchronizing multiple instruments (see :ACQuire:RSIGnal).

The :TIMebase:REFClock ON command enables the 10 MHz REF BNC and sets the reference signal mode to IN. The :TIMebase:REFClock OFF command disables the 10 MHz REF BNC (the same as setting the reference signal mode to OFF).

Query Syntax

```
:TIMebase:REFClock?
```

The :TIMebase:REFClock? query returns the current state of the 10 MHz reference signal mode. A "1" indicates that the 10 MHz REF input is enabled (on), and a "0" indicates that either the 10 MHz REF BNC is disabled (off) or that it is set as an output (by the :ACQuire:RSIGnal command).

Return Format

```
<value><NL>

<value> ::= {0 | 1}
```

See Also

- :ACQuire:RSIGnal

:TIMebase Commands

:TIMebase:REFerence — C

Command Syntax

```
:TIMebase:REFerence <reference>

<reference> ::= {LEFT | CENTer | RIGHt}
```

The :TIMebase:REFerence command sets the time reference to one division from the left side of the screen, to the center of the screen, or to one division from the right side of the screen. Time reference is the point on the display where the trigger point is referenced.

Query Syntax

```
:TIMebase:REFerence?
```

The :TIMebase:REFerence? query returns the current display reference for the main window.

Return Format

```
<reference><NL>

<reference> ::= {LEFT | CENT | RIGH}
```

See Also

- Introduction to :TIMebase Commands
- :TIMebase:MODE

Example Code

```
' TIME_REFERENCE - Possible values are LEFT and CENTER.
'  - LEFT sets the display reference on time division from the left.
'  - CENTER sets the display reference to the center of the screen.
myScope.WriteString ":TIMEBASE:REFERENCE CENTER"   ' Set reference to center.
```

Example program from the start: VISA COM Example in Visual Basic

:TIMebase Commands

:TIMebase:SCALe — N

Command Syntax

```
:TIMebase:SCALe <scale_value>

<scale_value> ::= 500 ps through 50 s in NR3 format
```

The :TIMebase:SCALe command sets the horizontal scale or units per division for the main window.

Query Syntax

```
:TIMebase:SCALe?
```

The :TIMebase:SCALe? query returns the current horizontal scale setting in seconds per division for the main window.

Return Format

```
<scale_value><NL>

<scale_value> ::= 500 ps through 50 s in NR3 format
```

See Also

- Introduction to :TIMebase Commands
- :TIMebase:RANGe
- :TIMebase:WINDow:SCALe
- :TIMebase:WINDow:RANGe

:TIMebase Commands

:TIMebase:VERNier — N

Command Syntax

```
:TIMebase:VERNier <vernier value>

<vernier value> ::= {{1 | ON} | {0 | OFF}
```

The :TIMebase:VERNier command specifies whether the time base control's vernier (fine horizontal adjustment) setting is ON (1) or OFF (0).

Query Syntax

```
:TIMebase:VERNier?
```

The :TIMebase:VERNier? query returns the current state of the time base control's vernier setting.

Return Format

```
<vernier value><NL>
```

```
<vernier value> ::= {0 | 1}
```

See Also

- Introduction to :TIMebase Commands

:TIMebase Commands

:TIMebase:WINDow:POSition — C

Command Syntax

```
:TIMebase:WINDow:POSition <pos value>
```

```
<pos value> ::= time from the trigger event to the delayed view reference point in NR3 format
```

The :TIMebase:WINDow:POSition command sets the horizontal position in the delayed view of the main sweep. The main sweep range and the main sweep horizontal position determine the range for this command. The value for this command must keep the delayed view window within the main sweep range.

Query Syntax

```
:TIMebase:WINDow:POSition?
```

The :TIMebase:WINDow:POSition? query returns the current horizontal window position setting in the delayed view.

Return Format

```
<value><NL>
```

```
<value> ::= position value in seconds
```

See Also

- Introduction to :TIMebase Commands
- :TIMebase:MODE
- :TIMebase:POSition
- :TIMebase:RANGe
- :TIMebase:SCALe
- :TIMebase:WINDow:RANGe
- :TIMebase:WINDow:SCALe

:TIMebase Commands

:TIMebase:WINDow:RANGe — C

Command Syntax

```
:TIMebase:WINDow:RANGe <range value>
```

```
<range value> ::= range value in seconds in NR3 format
```

The :TIMebase:WINDow:RANGe command sets the full-scale horizontal time in seconds for the delayed window. The range is 10 times the current delayed view window seconds per division setting. The main sweep range determines the range for this command. The maximum value is one half of the :TIMebase:RANGe value.

Query Syntax

```
:TIMebase:WINDow:RANGe?
```

The :TIMebase:WINDow:RANGe? query returns the current window timebase range setting.

Return Format

```
<value><NL>
```

```
<value> ::= range value in seconds
```

See Also

- Introduction to :TIMebase Commands
- :TIMebase:RANGe
- :TIMebase:POSition
- :TIMebase:SCALe

:TIMebase Commands

:TIMebase:WINDow:SCALe — N

Command Syntax

```
:TIMebase:WINDow:SCALe <scale_value>
```

```
<scale_value> ::= scale value in seconds in NR3 format
```

The :TIMebase:WINDow:SCALe command sets the delayed window horizontal scale (seconds/division). The main sweep scale determines the range for this command. The maximum value is one half of the :TIMebase:SCALe value.

Query Syntax

```
:TIMebase:WINDow:SCALe?
```

The :TIMebase:WINDow:SCALe? query returns the current delayed window scale setting.

Return Format

```
<scale_value><NL>

<scale_value> ::= current seconds per division for the delayed window
```

See Also

- Introduction to :TIMebase Commands
- :TIMebase:RANGe
- :TIMebase:POSition
- :TIMebase:SCALe
- :TIMebase:WINDow:RANGe

Commands by Subsystem

:TRIGger Commands

Control the trigger modes and parameters for each trigger type. See:

- Introduction to :TRIGger Commands
- General :TRIGger Commands
- :TRIGger:CAN Commands
- :TRIGger:DURation Commands
- :TRIGger:EBURst Commands
- :TRIGger[:EDGE] Commands
- :TRIGger:GLITch Commands (Pulse Width trigger)
- :TRIGger:IIC Commands
- :TRIGger:LIN Commands
- :TRIGger:SEQuence Commands
- :TRIGger:SPI Commands
- :TRIGger:TV Commands
- :TRIGger:USB Commands

Introduction to :TRIGger Commands

The commands in the TRIGger subsystem define the conditions for an internal trigger. Many of these commands are valid in multiple trigger modes.

The default trigger mode is :EDGE.

The trigger subsystem controls the trigger sweep mode and the trigger specification. The trigger sweep (see the :TRIGger:SWEep command) can be AUTO or NORMal.

- **NORMal** mode displays a waveform only if a trigger signal is present and the trigger conditions are met. Otherwise the oscilloscope does not trigger and the display is not updated. This mode is useful for low-repetitive-rate signals.
- **AUTO** trigger mode generates an artificial trigger event if the trigger specification is not satisfied within a preset time, acquires unsynchronized data and displays it.

  AUTO mode is useful for signals other than low-repetitive-rate signals. You must use this mode to display a DC signal because there are no edges on which to trigger.

The following trigger types are available (see the :TRIGger:MODE command).

- **CAN (Controller Area Network) triggering** will trigger on CAN version 2.0A and 2.0B signals. Setup consists of connecting the oscilloscope to a CAN signal. Baud rate, signal source, and signal polarity, and type of data to trigger on can be specified. With the automotive CAN and LIN serial decode option (Option ASM), you can also trigger on CAN data and identifier patterns, set the bit sample point, and have the module send an acknowledge to the bus when it receives a valid message.

  NOTE The CAN and LIN serial decode option (Option ASM) replaces the functionality that was available with the N2758A CAN trigger module for the 54620/54640 Series oscilloscopes.

- **Edge triggering** identifies a trigger by looking for a specified slope and voltage level on a waveform.
- **Nth Edge Burst triggering** lets you trigger on the Nth edge of a burst that occurs after an idle time.
- **Pulse width triggering** (:TRIGger:GLITch commands) sets the oscilloscope to trigger on a positive pulse or on a negative pulse of a specified width.
- **Pattern triggering** identifies a trigger condition by looking for a specified pattern. This pattern is a logical AND combination of the channels.
- **Duration triggering** lets you define a pattern, then trigger on a specified time duration.
- **IIC (Inter-IC bus) triggering** consists of connecting the oscilloscope to the serial data (SDA) line and the serial clock (SCL) line, then triggering on a stop/start condition, a restart, a missing acknowledge, or on a read/write frame with a specific device address and data value.
- **LIN (Local Interconnect Network) triggering** will trigger on LIN sync break at the beginning of a message frame. With the automotive CAN and LIN serial decode option (Option ASM), you can also trigger on Frame IDs.
- **Sequence triggering** allows you to trigger the oscilloscope after finding a sequence of events. Defining a sequence trigger requires three steps:
  1. Define the event to find before you trigger on the next event. This event can be a pattern, and edge from a single channel, or the combination of a pattern and a channel edge.
  2. Define the trigger event. This event can be a pattern, and edge from a single channel, the combination of a pattern and a channel edge, or the nth occurrence of an edge from a single channel.
  3. Set an optional reset event. This event can be a pattern, an edge from a single channel, the combination of a pattern and a channel edge, or a timeout value.
- **SPI (Serial Peripheral Interface) triggering** consists of connecting the oscilloscope to a clock, data, and framing signal. You can then trigger on a data pattern during a specific framing period. The serial data string can be specified to be from 4 to 32 bits long.
- **TV triggering** is used to capture the complicated waveforms of television equipment. The trigger circuitry detects the vertical and horizontal interval of the waveform and produces triggers based on the TV trigger settings you selected. TV triggering requires greater than ¼ division of sync amplitude with any analog channel as the trigger source.
- **USB (Universal Serial Bus) triggering** will trigger on a Start of Packet (SOP), End of Packet (EOP), Reset Complete, Enter Suspend, or Exit Suspend signal on the differential USB data lines. USB Low Speed and Full Speed are supported by this trigger.

Reporting the Setup. Use :TRIGger? to query setup information for the TRIGger subsystem.

Return Format. The return format for the TRIGger? query varies depending on the current mode. The following is a sample response from the :TRIGger? query. In this case, the query was issued following a *RST command.

```
:TRIG:MODE EDGE;SWE AUTO;NREJ 0;HFR 0;HOLD +60.0000000000000E-09;
:TRIG:EDGE:SOUR CHAN1;LEV +0.00000E+00;SLOP POS;REJ OFF;COUP DC
```

:TRIGger Commands

General :TRIGger Commands

Command	Query	Options and Query Returns
:TRIGger:HFReject {{0 | OFF} | {1 | ON}}	:TRIGger:HFReject?	{0 | 1}
:TRIGger:HOLDoff <holdoff_time>	:TRIGger:HOLDoff?	<holdoff_time> ::= 60 ns to 10 s in NR3 format

:TRIGger:MODE <mode>	:TRIGger:MODE?	<mode> ::= {EDGE | GLITch | PATTern | CAN | DURation | IIC | EBURst | LIN | SEQuence | SPI | TV | USB}
		<return_value> ::= {<mode> | <none>}
		<none> ::= query returns "NONE" if the :TIMebase:MODE is ROLL or XY

:TRIGger:NREJect {{0 | OFF} | {1 | ON}}	:TRIGger:NREJect?	{0 | 1}
:TRIGger:PATTern <value>, <mask> [,<edge source>,<edge>]	:TRIGger:PATTern?	<value> ::= 32-bit integer or <string> <mask> ::= 32-bit integer or <string> <string> ::= "0xnnnnnn"; n ::= {0,..,9 | A,..,F} <edge source> ::= {CHANnel<n> | EXTernal | NONE} for DSO models <edge source> ::= {CHANnel<n> | DIGital0,..,DIGital15 | NONE} for MSO models <edge> ::= {POSitive | NEGative} <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:SWEep <sweep>	:TRIGger:SWEep?	<sweep> ::= {AUTO | NORMal}

General :TRIGger Commands

:TRIGger:HFReject — C

Command Syntax

```
:TRIGger:HFReject <value>
```

```
<value> ::= {{0 | OFF} | {1 | ON}}
```

The :TRIGger:HFReject command turns the high frequency reject filter off and on. The high frequency reject filter adds a 50 kHz low-pass filter in the trigger path to remove high frequency components from the trigger waveform. Use this filter to remove high-frequency noise, such as AM or FM broadcast stations, from the trigger path.

Query Syntax

```
:TRIGger:HFReject?
```

The :TRIGger:HFReject? query returns the current high frequency reject filter mode.

Return Format

```
<value><NL>
```

```
<value> ::= {0 | 1}
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger[:EDGE]:REJect

General :TRIGger Commands

:TRIGger:HOLDoff — C

Command Syntax

```
:TRIGger:HOLDoff <holdoff_time>
```

```
<holdoff_time> ::= 60 ns to 10 s in NR3 format
```

The :TRIGger:HOLDoff command defines the holdoff time value in seconds. Holdoff keeps a trigger from occurring until after a certain amount of time has passed since the last trigger. This feature is valuable when a waveform crosses the trigger level multiple times during one period of the waveform. Without holdoff, the oscilloscope could trigger on each of the crossings, producing a confusing waveform. With holdoff set correctly, the oscilloscope always triggers on the same crossing. The correct holdoff setting is typically slightly less than one period.

Query Syntax

```
:TRIGger:HOLDoff?
```

The :TRIGger:HOLDoff? query returns the holdoff time value for the current trigger mode.

Return Format

```
<holdoff_time><NL>

<holdoff_time> ::= the holdoff time value in seconds in NR3 format.
```

See Also

- Introduction to :TRIGger Commands

General :TRIGger Commands

:TRIGger:MODE — C

Command Syntax

```
:TRIGger:MODE <mode>

<mode> ::= {EDGE | GLITch | PATTern | CAN | DURation | IIC | EBURst
| LIN | SEQuence | SPI | TV | USB}
```

The :TRIGger:MODE command selects the trigger mode (trigger type).

Query Syntax

```
:TRIGger:MODE?
```

The :TRIGger:MODE? query returns the current trigger mode. If the :TIMebase:MODE is ROLL or XY, the query returns "NONE."

Return Format

```
<mode><NL>

<mode> ::= {NONE | EDGE | GLITch | PATTern | CAN | DURation | IIC | EBURst
| LIN | SEQuence | SPI | TV | USB}
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:SWEep

Example Code

```
   ' TRIGGER_MODE - Set the trigger mode to EDGE, GLITch, PATTern, CAN,
   ' DURation, IIC, EBURSt, LIN, SEQuence, SPI, TV, or USB.
   myScope.WriteString ":TRIGGER:MODE EDGE"   ' Set the trigger mode to EDGE.
```

Example program from the start: VISA COM Example in Visual Basic

General :TRIGger Commands

:TRIGger:NREJect — C

Command Syntax

```
:TRIGger:NREJect <value>
```

```
<value> ::= {{0 | OFF} | {1 | ON}}
```

The :TRIGger:NREJect command turns the noise reject filter off and on. When the noise reject filter is on, the trigger circuitry is less sensitive to noise but may require a greater amplitude waveform to trigger the oscilloscope. This command is not valid in TV trigger mode.

Query Syntax

```
:TRIGger:NREJect?
```

The :TRIGger:NREJect? query returns the current noise reject filter mode.

Return Format

```
<value><NL>
```

```
<value> ::= {0 | 1}
```

See Also

- Introduction to :TRIGger Commands

General :TRIGger Commands

:TRIGger:PATTern — C

Command Syntax

```
:TRIGger: PATTern <pattern>
```

```
<pattern> ::= <value>, <mask> [, <edge source>, <edge>]
```

```
<value> ::= 32-bit integer or <string>
```

```
<mask> ::= 32-bit integer or <string>
```

```
<string> ::= "0xnnnnnn"; n ::= {0,..,9 | A,..,F}
```

```
<edge source> ::= {CHANnel<n> | EXTernal | NONE} for DSO models
```

```
<edge source> ::= {CHANnel<n> | DIGital0,..,DIGital15 | NONE} for MSO models
```

```
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
```

```
<n> ::= {1 | 2} for the two channel oscilloscope models
```

```
<edge> ::= {POSitive | NEGative}
```

The :TRIGger:PATTern command defines the specified pattern resource according to the value and the mask. For both<value> and <mask>, each bit corresponds to a possible trigger channel. The bit assignments vary by instrument:

Oscilloscope Models	Value and Mask Bit Assignments
4 analog + 16 digital channels (mixed-signal)	Bits 0 through 15 - digital channels 0 through 15. Bits 16 through 19 - analog channels 1 through 4.
2 analog + 16 digital channels (mixed-signal)	Bits 0 through 15 - digital channels 0 through 15. Bits 16 and 17 - analog channels 1 and 2.
4 analog channels only	Bits 0 through 3 - analog channels 1 through 4. Bit 4 - external trigger.
2 analog channels only	Bits 0 and 1 - analog channels 1 and 2. Bit 4 - external trigger.

Set a <value> bit to "0" to set the pattern for the corresponding channel to low. Set a <value> bit to "1" to set the pattern to high.

Set a <mask> bit to "0" to ignore the data for the corresponding channel. Only channels with a "1" set on the appropriate mask bit are used.

NOTE The optional source and the optional edge should be sent together or not at all. The edge will be set in the simple pattern if it is included. If the edge source is also specified in the mask, the edge takes precedence.

Query Syntax

```
:TRIGger:PATTern?
```

The :TRIGger:PATTern? query returns the pattern value, the mask, and the edge of interest in the simple pattern.

Return Format

```
<pattern><NL>
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:MODE

:TRIGger:SWEep — C

Command Syntax

```
:TRIGger:SWEep <sweep>
```

```
<sweep> ::= {AUTO | NORMal}
```

The :TRIGger:SWEep command selects the trigger sweep mode.

When AUTO sweep mode is selected, a baseline is displayed in the absence of a signal. If a signal is present but the oscilloscope is not triggered, the unsynchronized signal is displayed instead of a baseline.

When NORMal sweep mode is selected and no trigger is present, the instrument does not sweep, and the data acquired on the previous trigger remains on the screen.

NOTE This feature is called "Mode" on the instrument's front panel.

Query Syntax

```
:TRIGger:SWEep?
```

The :TRIGger:SWEep? query returns the current trigger sweep mode.

Return Format

```
<sweep><NL>

<sweep> ::= current trigger sweep mode
```

See Also

- Introduction to :TRIGger Commands

:TRIGger Commands

:TRIGger:CAN Commands

Command	Query	Options and Query Returns
:TRIGger:CAN:PATTern:DATA <value>, <mask>	:TRIGger:CAN:PATTern:DATA?	<value> ::= 64-bit integer in decimal, <nondecimal>, or <string> (with Option AMS) <mask> ::= 64-bit integer in decimal, <nondecimal>, or <string> <nondecimal> ::= #Hnn...n where n ::= {0,..,9 | A,..,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary

		<string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F} for hexadecimal
:TRIGger:CAN:PATTern:DATA:LENGth <length>	:TRIGger:CAN:PATTern:DATA:LENGth?	<length> ::= integer from 1 to 8 in NR1 format (with Option AMS)
:TRIGger:CAN:PATTern:ID <value>, <mask>	:TRIGger:CAN:PATTern:ID?	<value> ::= 32-bit integer in decimal, <nondecimal>, or <string> (with Option AMS) <mask> ::= 32-bit integer in decimal, <nondecimal>, or <string> <nondecimal> ::= #Hnn...n where n ::= {0,..,9 | A,..,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary <string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F} for hexadecimal
:TRIGger:CAN:PATTern:ID:MODE <value>	:TRIGger:CAN:PATTern:ID:MODE?	<value> ::= {STANdard | EXTended} (with Option AMS)
:TRIGger:CAN:SAMPlepoint <value>	:TRIGger:CAN:SAMPlepoint?	<value> ::= {60 | 62.5 | 68 | 70 | 75 | 80 | 87.5} in NR3 format
:TRIGger:CAN:SIGNal:BAUDrate <baudrate>	:TRIGger:CAN:SIGNal:BAUDrate?	<baudrate> ::= {10000 | 20000 | 33300 | 50000 | 62500 | 83300 | 100000 | 125000 | 250000 | 500000 | 800000 | 1000000}
:TRIGger:CAN:SOURce <source>	:TRIGger:CAN:SOURce?	<source> ::= {CHANnel<n> | EXTernal} for DSO models <source> ::= {CHANnel<n> | DIGital0,..,DIGital15 |} for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:CAN:TRIGger <condition>	:TRIGger:CAN:TRIGger?	<condition> ::= {SOF} (without Option AMS)

```
<condition> ::= {SOF |
DATA | ERRor | IDData |
IDEither | IDRemote |
ALLerrors | OVERload |
ACKerror} (with Option
AMS)
```

:TRIGger:CAN Commands

:TRIGger:CAN:PATTern:DATA — N

Command Syntax

```
:TRIGger:CAN:PATTern:DATA <value>,<mask>

<value> ::= 64-bit integer in decimal, <nondecimal>, or <string>

<mask> ::= 64-bit integer in decimal, <nondecimal>, or <string>

<nondecimal> ::= #Hnn...n where n ::= {0,..,9 | A,..,F} for hexadecimal

<nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary

<string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F} for hexadecimal
```

The :TRIGger:CAN:PATTern:DATA command defines the CAN data pattern resource according to the value and the mask. This pattern, along with the data length (set by the :TRIGger:CAN:PATTern:DATA:LENGth command), control the data pattern searched for in each CAN message.

Set a <value> bit to "0" to set the corresponding bit in the data pattern to low. Set a <value> bit to "1" to set the bit to high.

Set a <mask> bit to "0" to ignore that bit in the data stream. Only bits with a "1" set on the mask are used.

NOTE If more bytes are sent for <value> or <mask> than specified by the :TRIGger:CAN:PATTern:DATA:LENGth command, the most significant bytes will be truncated. If the data length is changed after the <value> and <mask> are programmed, the added or deleted bytes will be added to or deleted from the least significant bytes.

NOTE This command is only valid when the automotive CAN and LIN serial decode option (Option AMS) has been licensed.

Query Syntax

```
:TRIGger:CAN:PATTern:DATA?
```

The :TRIGger:CAN:PATTern:DATA? query returns the current settings of the specified CAN data pattern resource.

Return Format

```
<value>, <mask><NL> in nondecimal format
```

Errors

- Hardware missing

See Also

- Introduction to :TRIGger Commands
- :TRIGger:CAN:PATTern:DATA:LENGth
- :TRIGger:CAN:PATTern:ID

:TRIGger:CAN Commands

:TRIGger:CAN:PATTern:DATA:LENGth — N

Command Syntax

```
:TRIGger:CAN:PATTern:DATA:LENGth <length>

<length> ::= integer from 1 to 8 in NR1 format
```

The :TRIGger:CAN:PATTern:DATA:LENGth command sets the number of 8-bit bytes in the CAN data string. The number of bytes in the string can be anywhere from 0 bytes to 8 bytes (64 bits). The value for these bytes is set by the :TRIGger:CAN:PATTern:DATA command.

NOTE This command is only valid when the automotive CAN and LIN serial decode option (Option AMS) has been licensed.

Query Syntax

```
:TRIGger:CAN:PATTern:DATA:LENGth?
```

The :TRIGger:CAN:PATTern:DATA:LENGth? query returns the current CAN data pattern length setting.

Return Format

```
<count><NL>

<count> ::= integer from 1 to 8 in NR1 format
```

Errors

- Hardware missing

See Also

- Introduction to :TRIGger Commands
- :TRIGger:CAN:PATTern:DATA
- :TRIGger:CAN:SOURce

:TRIGger:CAN:PATTern:ID — N

Command Syntax

```
:TRIGger:CAN:PATTern:ID <value>, <mask>

<value> ::= 32-bit integer in decimal, <nondecimal>, or <string>

<mask> ::= 32-bit integer in decimal, <nondecimal>, or <string>

<nondecimal> ::= #Hnn...n where n ::= {0,..,9 | A,..,F} for hexadecimal

<nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary

<string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F} for hexadecimal
```

The :TRIGger:CAN:PATTern:ID command defines the CAN identifier pattern resource according to the value and the mask. This pattern, along with the identifier mode (set by the :TRIGger:CAN:PATTern:ID:MODE command), control the identifier pattern searched for in each CAN message.

Set a <value> bit to "0" to set the corresponding bit in the identifier pattern to low. Set a <value> bit to "1" to set the bit to high.

Set a <mask> bit to "0" to ignore that bit in the identifier stream. Only bits with a "1" set on the mask are used.

NOTE If more bits are sent than allowed (11 bits in standard mode, 29 bits in extended mode) by the :TRIGger:CAN:PATTern:ID:MODE command, the most significant bytes will be truncated. If the ID mode is changed after the <value> and <mask> are programmed, the added or deleted bits will be added to or deleted from the most significant bits.

NOTE This command is only valid when the automotive CAN and LIN serial decode option (Option AMS) has been licensed.

Query Syntax

```
:TRIGger:CAN:PATTern:ID?
```

The :TRIGger:CAN:PATTern:ID? query returns the current settings of the specified CAN identifier pattern resource.

Return Format

```
<value>, <mask><NL> in nondecimal format
```

Errors

- Hardware missing

See Also

- Introduction to :TRIGger Commands
- :TRIGger:CAN:PATTern:ID:MODE
- :TRIGger:CAN:PATTern:DATA

:TRIGger:CAN Commands

:TRIGger:CAN:PATTern:ID:MODE — N

Command Syntax

```
:TRIGger:CAN:PATTern:ID:MODE <value>

<value> ::= {STANdard | EXTended}
```

The :TRIGger:CAN:PATTern:ID:MODE command sets the CAN identifier mode. STANdard selects the standard 11-bit identifier. EXTended selects the extended 29-bit identifier. The CAN identifier is set by the :TRIGger:CAN:PATTern:ID command.

NOTE This command is only valid when the automotive CAN and LIN serial decode option (Option AMS) has been licensed.

Query Syntax

```
:TRIGger:CAN:PATTern:ID:MODE?
```

The :TRIGger:CAN:PATTern:ID:MODE? query returns the current setting of the CAN identifier mode.

Return Format

```
<value><NL>

<value> ::= {STAN | EXT}
```

Errors

- Hardware missing

See Also

- Introduction to :TRIGger Commands
- :TRIGger:MODE
- :TRIGger:CAN:PATTern:DATA
- :TRIGger:CAN:PATTern:DATA:LENGth
- :TRIGger:CAN:PATTern:ID

:TRIGger:CAN Commands

:TRIGger:CAN:SAMPlepoint — N

Command Syntax

```
:TRIGger:CAN:SAMPlepoint <value>
```

```
<value><NL>
```

```
<value> ::= {60 | 62.5 | 68 | 70 | 75 | 80 | 87.5} in NR3 format
```

The :TRIGger:CAN:SAMPlepoint command sets the point during the bit time where the bit level is sampled to determine whether the bit is dominant or recessive. The sample point represents the percentage of time between the beginning of the bit time to the end of the bit time.

Query Syntax

```
:TRIGger:CAN:SAMPlepoint?
```

The :TRIGger:CAN:SAMPlepoint? query returns the current CAN sample point setting.

Return Format

```
<value><NL>
```

```
<value> ::= {60 | 62.5 | 68 | 70 | 75 | 80 | 87.5} in NR3 format
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:MODE
- :TRIGger:CAN:TRIGger

:TRIGger:CAN Commands

:TRIGger:CAN:SIGNal:BAUDrate — N

Command Syntax

```
:TRIGger:CAN:SIGNal:BAUDrate <baudrate>
```

```
<baudrate> ::= integer in NR1 format
```

```
<baudrate> ::= {10000 | 20000 | 33300 | 50000 | 62500 | 83300 | 100000
| 125000 | 250000 | 500000 | 800000 |1000000}
```

The :TRIGger:CAN:SIGNal:BAUDrate command sets the standard baud rate of the CAN signal from 10 kb/s to 1 Mb/s. If a non-standard baud rate is sent, the baud rate will be set to the next highest standard rate.

If the baud rate you select does not match the system baud rate, false triggers may occur.

Query Syntax

```
:TRIGger:CAN:SIGNal:BAUDrate?
```

The :TRIGger:CAN:SIGNal:BAUDrate? query returns the current CAN baud rate setting.

Return Format

```
<baudrate><NL>
```

```
<baudrate> ::= integer = {10000 | 20000 | 33300 | 50000 | 62500 | 83300 |
100000 | 125000 | 250000 | 500000 | 800000 |1000000}
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:MODE
- :TRIGger:CAN:TRIGger
- :TRIGger:CAN:SIGNal:DEFinition
- :TRIGger:CAN:SOURce

:TRIGger:CAN Commands

:TRIGger:CAN:SOURce — N

Command Syntax

```
:TRIGger:CAN:SOURce <source>
```

```
<source> ::= {CHANnel<n> | EXTernal} for the DSO models
```

```
<source> ::= {CHANnel<n> | DIGital0,..,DIGital15} for the MSO models
```

```
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
```

```
<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :TRIGger:CAN:SOURce command sets the source for the CAN signal. The source setting is only valid when :TRIGger:CAN:TRIGger is set to SOF (start of frame).

Query Syntax

```
:TRIGger:CAN:SOURce?
```

The :TRIGger:CAN:SOURce? query returns the current source for the CAN signal.

Return Format

```
<source><NL>
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:MODE
- :TRIGger:CAN:TRIGger
- :TRIGger:CAN:SIGNal:DEFinition

:TRIGger:CAN Commands

:TRIGger:CAN:TRIGger — N

Command Syntax

```
:TRIGger:CAN:TRIGger <condition>

<condition> ::= {SOF | DATA | ERRor | IDData | IDEither | IDRemote |
                ALLerrors | OVERload | ACKerror}
```

The :TRIGger:CAN:TRIGger command sets the CAN trigger on condition:

- SOF - will trigger on the Start of Frame (SOF) bit of a Data frame, Remote Transfer Request (RTR) frame, or an Overload frame.
- DATA - will trigger on CAN Data frames matching the specified Id, Data, and the DLC (Data length code).
- ERRor - will trigger on CAN Error frame.
- IDData - will trigger on CAN frames matching the specified Id of a Data frame.
- IDEither - will trigger on the specified Id, regardless if it is a Remote frame or a Data frame.
- IDRemote - will trigger on CAN frames matching the specified Id of a Remote frame.
- ALLerrors - will trigger on CAN active error frames and unknown bus conditions.
- OVERload - will trigger on CAN overload frames.
- ACKerror - will trigger on a data or remote frame acknowledge bit that is recessive.

The table below shows the programming parameter and the corresponding front-panel softkey selection:

Remote <condition> parameter	Front-panel Trigger on: softkey selection (softkey text - softkey popup text)
SOF	SOF - Start of Frame
DATA	Id & Data - Data Frame Id and Data
ERRor	Error - Error frame
IDData	Id & ~RTR - Data Frame Id (~RTR)
IDEither	Id - Remote or Data Frame Id
IDRemote	Id & RTR - Remote Frame Id (RTR)
ALLerrors	All Errors - All Errors

OVERload	Overload - Overload Frame
ACKerror	Ack Error - Acknowledge Error

CAN Id specification is set by the :TRIGger:CAN:PATTern:ID and:TRIGger:CAN:PATTern:ID:MODE commands.

CAN Data specification is set by the :TRIGger:CAN:PATTern:DATA command.

CAN Data Length Code is set by the :TRIGger:CAN:PATTern:DATA:LENGth command.

NOTE SOF is the only valid selection for analog oscilloscopes. If the automotive CAN and LIN serial decode option (Option AMS) has not been licensed, SOF is the only valid selection.

Query Syntax

```
:TRIGger:CAN:TRIGger?
```

The :TRIGger:CAN:TRIGger? query returns the current CAN trigger on condition.

Return Format

```
<condition><NL>

<condition> ::= {SOF | DATA | ERR | IDD | IDE | IDR | ALL | OVER | ACK}
```

Errors

- Hardware missing

See Also

- Introduction to :TRIGger Commands
- :TRIGger:MODE
- :TRIGger:CAN:PATTern:DATA
- :TRIGger:CAN:PATTern:DATA:LENGth
- :TRIGger:CAN:PATTern:ID
- :TRIGger:CAN:PATTern:ID:MODE
- :TRIGger:CAN:SIGNal:DEFinition
- :TRIGger:CAN:SOURce

:TRIGger Commands

:TRIGger:DURation Commands

Command	Query	Options and Query Returns
:TRIGger:DURation:GREaterthan	:TRIGger:DURation:GREaterthan?	<greater than time> ::=

<greater than time>[suffix]		floating-point number from 5 ns to 10 seconds in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:DURation:LESSthan <less than time>[suffix]	:TRIGger:DURation:LESSthan?	<less than time> ::= floating-point number from 5 ns to 10 seconds in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:DURation:PATTern <value>, <mask>	:TRIGger:DURation:PATTern?	<value> ::= integer or <string> <mask> ::= integer or <string> <string> ::= ""0xnnnnnn"" n ::= {0,..,9 | A,..,F}
:TRIGger:DURation:QUALifier <qualifier>	:TRIGger:DURation:QUALifier?	<qualifier> ::= {GREaterthan | LESSthan | INRange | OUTRange | TIMeout}
:TRIGger:DURation:RANGe <greater than time>[suffix], <less than time>[suffix]	:TRIGger:DURation:RANGe?	<greater than time> ::= min duration from 10 ns to 9.99 seconds in NR3 format <less than time> ::= max duration from 15 ns to 10 seconds in NR3 format [suffix] ::= {s | ms | us | ns | ps}

:TRIGger:DURation Commands

:TRIGger:DURation:GREaterthan — N

Command Syntax

```
:TRIGger:DURation:GREaterthan <greater than time>[<suffix>]

<greater than time> ::= minimum trigger duration in seconds (5 ns - 10 seconds) in NR3 format

<suffix> ::= {s | ms | us | ns | ps }
```

The :TRIGger:DURation:GREaterthan command sets the minimum duration for the defined pattern when :TRIGger:DURation:QUALifier is set to GREaterthan. The command also sets the timeout value when the :TRIGger:DURation:QUALifier is set to TIMeout.

Query Syntax

```
:TRIGger:DURation:GREaterthan?
```

The :TRIGger:DURation:GREaterthan? query returns the minimum duration time for the defined pattern.

Return Format

```
<greater than time><NL>
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:DURation:PATTern
- :TRIGger:DURation:QUALifier
- :TRIGger:MODE

:TRIGger:DURation Commands

:TRIGger:DURation:LESSthan — N

Command Syntax

```
:TRIGger:DURation:LESSthan <less than time>[<suffix>]

<less than time> ::= maximum trigger duration in seconds (5 ns - 10 seconds) in NR3 format

<suffix> ::= {s | ms | us | ns | ps}
```

The :TRIGger:DURation:LESSthan command sets the maximum duration for the defined pattern when :TRIGger:DURation:QUALifier is set to LESSthan.

Query Syntax

```
:TRIGger:DURation:LESSthan?
```

The :TRIGger:DURation:LESSthan? query returns the duration time for the defined pattern.

Return Format

```
<less than time><NL>
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:DURation:PATTern
- :TRIGger:DURation:QUALifier
- :TRIGger:MODE

:TRIGger:DURation:PATTern — N

Command Syntax

```
:TRIGger:DURation:PATTern <value>, <mask>

<value> ::= integer or<string>

<mask> ::= integer or <string>

<string> ::= "0xnnnnnnn"; n ::= {0,..,9 | A,..,F}
```

The :TRIGger:DURation:PATTern command defines the specified duration pattern resource according to the value and the mask. For both <value> and <mask>, each bit corresponds to a possible trigger channel. The bit assignments vary by instrument:

Oscilloscope Models	Value and Mask Bit Assignments
4 analog + 16 digital channels (mixed-signal)	Bits 0 through 15 - digital channels 0 through 15. Bits 16 through 19 - analog channels 1 through 4.
2 analog + 16 digital channels (mixed-signal)	Bits 0 through 15 - digital channels 0 through 15. Bits 16 and 17 - analog channels 1 and 2.
4 analog channels only	Bits 0 through 3 - analog channels 1 through 4. Bit 4 - external trigger.
2 analog channels only	Bits 0 and 1 - analog channels 1 and 2. Bit 4 - external trigger.

Set a <value> bit to "0" to set the pattern for the corresponding channel to low. Set a <value> bit to "1" to set the pattern to high.

Set a <mask> bit to "0" to ignore the data for the corresponding channel. Only channels with a "1" set on the appropriate mask bit are used.

Query Syntax

```
:TRIGger:DURation:PATTern?
```

The :TRIGger:DURation:PATTern? query returns the pattern value.

Return Format

```
<value>, <mask><NL>

<value> ::= a 32-bit integer in NR1 format.

<mask> ::= a 32-bit integer in NR1 format.
```

See Also

- Introduction to :TRIGger Commands

- :TRIGger:PATTern

:TRIGger:DURation Commands

:TRIGger:DURation:QUALifier — N

Command Syntax

```
:TRIGger:DURation:QUALifier <qualifier>

<qualifier> ::= {GREaterthan | LESSthan | INRange | OUTRange | TIMeout}
```

The :TRIGger:DURation:QUALifier command qualifies the trigger duration.

Set the GREaterthan qualifier value with the :TRIGger:DURation:GREaterthan command.

Set the LESSthan qualifier value with the :TRIGger:DURation:LESSthan command.

Set the INRange and OUTRange qualifier values with the :TRIGger:DURation:RANGe command.

Set the TIMeout qualifier value with the :TRIGger:DURation:GREaterthan command.

Query Syntax

```
:TRIGger:DURation:QUALifier?
```

The :TRIGger:DURation:QUALifier? query returns the trigger duration qualifier.

Return Format

```
<qualifier><NL>
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:DURation:GREaterthan
- :TRIGger:DURation:LESSthan
- :TRIGger:DURation:RANGe

:TRIGger:DURation Commands

:TRIGger:DURation:RANGe — N

Command Syntax

```
:TRIGger:DURation:RANGe <greater than time>[<suffix>],<less than time>[<suffix>]

<greater than time> ::= minimum duration in seconds (10 ns - 9.99 seconds) in NR3 format
```

```
<less than time> ::= maximum duration in seconds (15 ns - 10 seconds) in NR3 format

<suffix> ::= {s | ms | us | ns | ps}
```

The :TRIGger:DURation:RANGe command sets the duration for the defined pattern when the :TRIGger:DURation:QUALifier command is set to INRange or OUTRange.

NOTE If you set the minimum duration longer than the maximum duration, the order of the parameters is automatically reversed.

Query Syntax

```
:TRIGger:DURation:RANGe?
```

The :TRIGger:DURation:RANGe? query returns the duration time for the defined pattern.

Return Format

```
<greater than time>,<less than time><NL>
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:DURation:PATTern
- :TRIGger:DURation:QUALifier
- :TRIGger:MODE

:TRIGger Commands

:TRIGger:EBURst Commands

Command	Query	Options and Query Returns
:TRIGger:EBURst:COUNt <count>	:TRIGger:EBURst:COUNt?	<count> ::= integer in NR1 format
:TRIGger:EBURst:IDLE <time_value>	:TRIGger:EBURst:IDLE?	<time_value> ::= time in seconds in NR3 format
:TRIGger:EBURst:SLOPe <slope>	:TRIGger:EBURst:SLOPe?	<slope> ::= {NEGative | POSitive}

The :TRIGger[:EDGE]:SOURce command is used to specify the source channel for the Nth Edge Burst trigger. If an analog channel is selected as the source, the :TRIGger[:EDGE]:LEVel command is used to set the Nth Edge Burst trigger level. If a digital channel is selected as the source, the :DIGital<n>:THReshold or :POD<n>:THReshold command is used to set the Nth Edge Burst trigger level.

:TRIGger:EBURst Commands

:TRIGger:EBURst:COUNt — N

Command Syntax

```
:TRIGger:EBURst:COUNt <count>
```

```
<count> ::= integer in NR1 format
```

The :TRIGger:EBURst:COUNt command sets the Nth edge at burst counter resource. The edge counter is used in the trigger stage to determine which edge in a burst will generate a trigger.

Query Syntax

```
:TRIGger:EBURst:COUNt?
```

The :TRIGger:EBURst:COUNt? query returns the current Nth edge of burst edge counter setting.

Return Format

```
<count><NL>
```

```
<count> ::= integer in NR1 format
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:EBURst:SLOPe
- :TRIGger:EBURst:IDLE

:TRIGger:EBURst Commands

:TRIGger:EBURst:IDLE — N

Command Syntax

```
:TRIGger:EBURst:IDLE <time_value>
```

```
<time_value> ::= time in seconds in NR3 format
```

The :TRIGger:EBURst:IDLE command sets the Nth edge in a burst idle resource in seconds from 10 ns to 10 s. The timer is used to set the minimum time before the next burst.

Query Syntax

```
:TRIGger:EBURst:IDLE?
```

The :TRIGger:EBURst:IDLE? query returns current Nth edge in a burst idle setting.

Return Format

```
<time value><NL>
```

```
<time_value> ::= time in seconds in NR3 format
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:EBURst:SLOPe
- :TRIGger:EBURst:COUNt

:TRIGger:EBURst Commands

:TRIGger:EBURst:SLOPe — N

Command Syntax

```
:TRIGger:EBURst:SLOPe <slope>

<slope> ::= {NEGative | POSitive}
```

The :TRIGger:EBURst:SLOPe command specifies whether the rising edge (POSitive) or falling edge (NEGative) of the Nth edge in a burst will generate a trigger.

Query Syntax

```
:TRIGger:EBURst:SLOPe?
```

The :TRIGger:EBURst:SLOPe? query returns the current Nth edge in a burst slope.

Return Format

```
<slope><NL>

<slope> ::= {NEG | POS}
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:EBURst:IDLE
- :TRIGger:EBURst:COUNt

:TRIGger Commands

:TRIGger[:EDGE] Commands

Command	Query	Options and Query Returns
:TRIGger[:EDGE]:COUPling {AC | DC | LF}	:TRIGger[:EDGE]:COUPling?	{AC | DC | LF}
:TRIGger[:EDGE]:LEVel <level> [,<source>]	:TRIGger[:EDGE]:LEVel? [<source>]	For internal triggers, <level> ::= .75 x full-scale voltage from center screen in NR3 format. For external triggers, <level> ::= 2 volts with probe attenuation at 1:1 in NR3

		format. For digital channels (MSO models), <level> ::= 8 V. <source> ::= {CHANnel<n> | EXTernal} for DSO models <source> ::= {CHANnel<n> | DIGital0,..,DIGital15 | EXTernal } for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger[:EDGE]:REJect {OFF | LF | HF}	:TRIGger [:EDGE]:REJect?	{OFF | LF | HF}
:TRIGger[:EDGE]:SLOPe <polarity>	:TRIGger [:EDGE]:SLOPe?	<polarity> ::= {POSitive | NEGative | EITHer | ALTernate}
:TRIGger[:EDGE]:SOURce <source>	:TRIGger [:EDGE]:SOURce?	<source> ::= {CHANnel<n> | EXTernal} for DSO models <source> ::= {CHANnel<n> | DIGital0,..,DIGital15 | EXTernal} for MSO models <n> ::= 1-2 or 1-4 in NR1 format

:TRIGger[:EDGE] Commands

:TRIGger[:EDGE]:COUPling — C

Command Syntax

```
:TRIGger[:EDGE]:COUPling <coupling>

<coupling> ::= {AC | DC | LFReject}
```

The :TRIGger[:EDGE]:COUPling command sets the input coupling for the selected trigger sources. The coupling can be set to AC, DC, or LF. AC coupling places a high-pass filter (10 Hz for analog channels, and 3.5 Hz for all External trigger inputs) in the trigger path, removing dc offset voltage from the trigger waveform. LF coupling places a 50 KHz high-pass filter in the trigger path. Use AC coupling to get a stable edge trigger when your waveform has a large dc offset. DC coupling allows dc and ac signals into the trigger path.

NOTE The :TRIGger[:EDGE]:COUPling and the :TRIGger[:EDGE]:REJect selections are coupled. Changing the setting of the :TRIGger[:EDGE]:REJect can change the COUPling setting.

Query Syntax

```
:TRIGger[:EDGE]:COUPling?
```

The :TRIGger[:EDGE]:COUPling? query returns the current coupling selection.

Return Format

```
<coupling><NL>

<coupling> ::= {AC | DC | LFR}
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:MODE
- :TRIGger[:EDGE]:REJect

:TRIGger[:EDGE] Commands

:TRIGger[:EDGE]:LEVel — C

Command Syntax

```
:TRIGger[:EDGE]:LEVel <level>

<level> ::= <level>[,<source>]

<level> ::= 0.75 x full-scale voltage from center screen in NR3 format for internal triggers

<level> ::= 2 V with probe attenuation at 1:1 in NR3 format for external triggers

<level> ::= 8 V for digital channels (MSO models)

<source> ::= {CHANnel<n> | EXTernal} for the DSO models

<source> ::= {CHANnel<n> | DIGital0,..,DIGital15 | EXTernal} for the MSO models

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :TRIGger[:EDGE]:LEVel command sets the trigger level voltage for the active trigger source.

NOTE If the optional source is specified and is not the active source, the level on the active source is not affected and the active source is not changed.

Query Syntax

```
:TRIGger[:EDGE]:LEVel? [<source>]
```

The :TRIGger[:EDGE]:LEVel? query returns the trigger level of the current trigger source.

Return Format

```
<level><NL>
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger[:EDGE]:SOURce
- :POD<n>:THReshold
- :DIGital<n>:THReshold

:TRIGger[:EDGE] Commands

:TRIGger[:EDGE]:REJect — C

Command Syntax

```
:TRIGger[:EDGE]:REJect <reject>

<reject> ::= {OFF | LFReject | HFReject}
```

The :TRIGger[:EDGE]:REJect command turns the low-frequency or high-frequency reject filter on or off. You can turn on one of these filters at a time. The high frequency reject filter adds a 50 kHz low-pass filter in the trigger path to remove high frequency components from the trigger waveform. Use the high frequency reject filter to remove high-frequency noise, such as AM or FM broadcast stations, from the trigger path. The low frequency reject filter adds a 50 kHz high-pass filter in series with the trigger waveform to remove any unwanted low frequency components from a trigger waveform, such as power line frequencies, that can interfere with proper triggering.

NOTE The :TRIGger[:EDGE]:REJect and the :TRIGger[:EDGE]:COUPling selections are coupled. Changing the setting of the :TRIGger[:EDGE]:COUPling can change the COUPling setting.

Query Syntax

```
:TRIGger[:EDGE]:REJect?
```

The :TRIGger[:EDGE]:REJect? query returns the current status of the reject filter.

Return Format

```
<reject><NL>

<reject> ::= {OFF | LFR | HFR}
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:HFReject
- :TRIGger[:EDGE]:COUPling

:TRIGger[:EDGE] Commands

:TRIGger[:EDGE]:SLOPe — C

Command Syntax

```
:TRIGger[:EDGE]:SLOPe <slope>

<slope> ::= {NEGative | POSitive | EITHer | ALTernate}
```

The :TRIGger[:EDGE]:SLOPe command specifies the slope of the edge for the trigger. The SLOPe command is not valid in TV trigger mode. Instead, use :TRIGger:TV:POLarity to set the polarity in TV trigger mode.

Query Syntax

```
:TRIGger[:EDGE]:SLOPe?
```

The :TRIGger[:EDGE]:SLOPe? query returns the current trigger slope.

Return Format

```
<slope><NL>

<slope> ::= {NEG | POS | EITH | ALT}
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:MODE
- :TRIGger:TV:POLarity

Example Code

```
   ' TRIGGER_EDGE_SLOPE - Sets the slope of the edge for the trigger.
   myScope.WriteString ":TRIGGER:EDGE:SLOPE POSITIVE"   ' Set the slope to positive.
```

Example program from the start: VISA COM Example in Visual Basic

:TRIGger[:EDGE] Commands

:TRIGger[:EDGE]:SOURce — C

Command Syntax

```
:TRIGger[:EDGE]:SOURce <source>

<source> ::= {CHANnel<n> | EXTernal | LINE} for the DSO models

<source> ::= {CHANnel<n> | DIGital0,..,DIGital15 | EXTernal | LINE} for the MSO models

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :TRIGger[:EDGE]:SOURce command selects the channel that produces the trigger.

Query Syntax

```
:TRIGger[:EDGE]:SOURce?
```

The :TRIGger[:EDGE]:SOURce? query returns the current source. If all channels are off, the query returns "NONE."

Return Format

```
<source><NL>

<source> ::= {CHAN<n> | EXT | LINE | NONE} for the DSO models

<source> ::= {CHAN<n> | DIG0,..,DIG15 | EXTernal | LINE | NONE} for the MSO models
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:MODE

Example Code

```
   ' TRIGGER_TV_SOURCE - Selects the channel that actually produces the
   ' TV trigger.  Any channel can be selected.
   myScope.WriteString ":TRIGGER:TV:SOURCE CHANNEL1"
```

Example program from the start: VISA COM Example in Visual Basic

:TRIGger Commands

:TRIGger:GLITch Commands

Command	Query	Options and Query Returns
:TRIGger:GLITch:GREaterthan <greater than time>[suffix]	:TRIGger:GLITch:GREaterthan?	<greater than time> ::= floating-point number from 5 ns to 10 seconds in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:GLITch:LESSthan <less than time>[suffix]	:TRIGger:GLITch:LESSthan?	<less than time> ::= floating-point number from 5 ns to 10 seconds in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:GLITch:LEVel <level> [<source>]	:TRIGger:GLITch:LEVel?	For internal triggers, <level> ::= .75 x full-scale voltage from center screen in NR3 format.

		For external triggers, <level> ::= 2 volts with probe attenuation at 1:1 in NR3 format. For digital channels (MSO models), <level> ::= 6 V. <source> ::= {CHANnel<n> | EXTernal} for DSO models <source> ::= {CHANnel<n> | DIGital0,..,DIGital15} for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:GLITch:POLarity <polarity>	:TRIGger:GLITch:POLarity?	<polarity> ::= {POSitive | NEGative}
:TRIGger:GLITch:QUALifier <qualifier>	:TRIGger:GLITch:QUALifier?	<qualifier> ::= {GREaterthan | LESSthan | RANGe}
:TRIGger:GLITch:RANGE <greater than time>[suffix], <less than time>[suffix]	:TRIGger:GLITch:RANGe?	<greater than time> ::= start time from 10 ns to 9.99 seconds in NR3 format <less than time> ::= stop time from 15 ns to 10 seconds in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:GLITch:SOURce <source>	:TRIGger:GLITch:SOURce?	<source> ::= {CHANnel<n> | EXTernal} for DSO models <source> ::= {CHANnel<n> | DIGital0,..,DIGital15 } for MSO models <n> ::= 1-2 or 1-4 in NR1 format

:TRIGger:GLITch Commands

:TRIGger:GLITch:GREaterthan — N

Command Syntax

```
:TRIGger:GLITch:GREaterthan <greater_than_time>[<suffix>]

<greater_than_time> ::= 32-bit floating-point number (5 ns - 10 seconds) in NR3 format

<suffix> ::= {s | ms | us | ns | ps}
```

The :TRIGger:GLITch:GREaterthan command sets the minimum pulse width duration for the selected :TRIGger:GLITch:SOURce.

Query Syntax

```
:TRIGger:GLITch:GREaterthan?
```

The :TRIGger:GLITch:GREaterthan? query returns the minimum pulse width duration time for :TRIGger:GLITch:SOURce.

Return Format

```
<greater_than_time><NL>.
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:GLITch:SOURce
- :TRIGger:GLITch:QUALifier
- :TRIGger:MODE

:TRIGger:GLITch Commands

:TRIGger:GLITch:LESSthan — N

Command Syntax

```
:TRIGger:GLITch:LESSthan <less_than_time>[<suffix>]

<less_than_time> ::= floating-point number (5 ns - 10 seconds)

<suffix> ::= {s | ms | us | ns | ps}
```

The :TRIGger:GLITch:LESSthan command sets the maximum pulse width duration for the selected :TRIGger:GLITch:SOURce.

Query Syntax

```
:TRIGger:GLITch:LESSthan?
```

The :TRIGger:GLITch:LESSthan? query returns the pulse width duration time for :TRIGger:GLITch:SOURce.

Return Format

```
<less_than_time><NL>

<less_than_time> ::= a 32-bit floating-point number in NR3 format.
```

See Also

- Introduction to :TRIGger Commands

- :TRIGger:GLITch:SOURce
- :TRIGger:GLITch:QUALifier
- :TRIGger:MODE

:TRIGger:GLITch Commands

:TRIGger:GLITch:LEVel — N

Command Syntax

```
:TRIGger:GLITch:LEVel <level_argument>

<level_argument> ::= <level>[, <source>]

<level> ::= .75 x full-scale voltage from center screen in NR3 format for internal triggers

<level> ::= 2 V with probe attenuation at 1:1 in NR3 format for external triggers

<level> ::= 6 V for digital channels (MSO models)

<source> ::= {CHANnel<n> | EXTernal} for DSO models

<source> ::= {CHANnel<n> | DIGital0,..,DIGital15} for MSO models

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :TRIGger:GLITch:LEVel command sets the trigger level voltage for the active pulse width trigger.

Query Syntax

```
:TRIGger:GLITch:LEVel?
```

The :TRIGger:GLITch:LEVel? query returns the trigger level of the current pulse width trigger mode. If all channels are off, the query returns "NONE."

Return Format

```
<level_argument><NL>
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:MODE
- :TRIGger:GLITch:SOURce

:TRIGger:GLITch Commands

:TRIGger:GLITch:POLarity — N

Command Syntax

```
:TRIGger:GLITch:POLarity <polarity>
```

```
<polarity> ::= {POSitive | NEGative}
```

The :TRIGger:GLITch:POLarity command sets the polarity for the glitch pulse width trigger.

Query Syntax

```
:TRIGger:GLITch:POLarity?
```

The :TRIGger:GLITch:POLarity? query returns the glitch pulse width trigger polarity.

Return Format

```
<polarity><NL>
```

```
<polarity> ::= {POS | NEG}
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:MODE
- :TRIGger:GLITch:SOURce

:TRIGger:GLITch Commands

:TRIGger:GLITch:QUALifier — N

Command Syntax

```
:TRIGger:GLITch:QUALifier <operator>
```

```
<operator> ::= {GREaterthan | LESSthan | RANGe}
```

This command sets the mode of operation of the glitch pulse width trigger. The oscilloscope can trigger on a pulse width that is greater than a time value, less than a time value, or within a range of time values.

Query Syntax

```
:TRIGger:GLITch:QUALifier?
```

The :TRIGger:GLITch:QUALifier? query returns the glitch pulse width qualifier.

Return Format

```
<operator><NL>
```

```
<operator> ::= {GRE | LESS | RANG}
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:GLITch:SOURce
- :TRIGger:MODE

:TRIGger:GLITch Commands

:TRIGger:GLITch:RANGe — N

Command Syntax

```
:TRIGger:GLITch:RANGe <greater than time>[suffix],<less than time>[suffix]

<greater than time> ::= start time (10 ns - 9.99 seconds) in NR3 format

<less than time> ::= stop time (15 ns - 10 seconds) in NR3 format

[suffix] ::= {s | ms | us | ns | ps}
```

The :TRIGger:GLITch:RANGe command sets the pulse width duration for the selected :TRIGger:GLITch:SOURce. If you set the stop time before the start time, the order of the parameters is automatically reversed.

Query Syntax

```
:TRIGger:GLITch:RANGe?
```

The :TRIGger:GLITch:RANGe? query returns the pulse width duration time for :TRIGger:GLITch:SOURce.

Return Format

```
<start time>,<stop time><NL>
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:GLITch:SOURce
- :TRIGger:GLITch:QUALifier
- :TRIGger:MODE

:TRIGger:GLITch Commands

:TRIGger:GLITch:SOURce — N

Command Syntax

```
:TRIGger:GLITch:SOURce <source>

<source> ::= {CHANnel<n> | EXTernal} for the DSO models

<source> ::= {DIGital0,..,DIGital15 | CHANnel<n>} for the MSO models

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :TRIGger:GLITch:SOURce command selects the channel that produces the pulse width trigger.

Query Syntax

```
:TRIGger:GLITch:SOURce?
```

The :TRIGger:GLITch:SOURce? query returns the current pulse width source. If all channels are off, the query returns "NONE."

Return Format

```
<source><NL>
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:MODE
- :TRIGger:GLITch:LEVel
- :TRIGger:GLITch:POLarity
- :TRIGger:GLITch:QUALifier
- :TRIGger:GLITch:RANGe

Example Code

- Example Code

:TRIGger Commands

:TRIGger:IIC Commands

Command	Query	Options and Query Returns
:TRIGger:IIC:PATTern:ADDRess <value>	:TRIGger:IIC:PATTern:ADDRess?	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,..,9 | A,..,F}
:TRIGger:IIC:PATTern:DATA <value>	:TRIGger:IIC:PATTern:DATA?	<value> ::= integer or <string> <string> ::= "0xnn" n ::=

		{0,..,9 | A,..,F}
:TRIGger:IIC:PATTern:DATa2 <value>	:TRIGger:IIC:PATTern:DATa2?	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,..,9 | A,..,F}
:TRIGger:IIC[:SOURce]:CLOCk <source>	:TRIGger:IIC[:SOURce]:CLOCk?	<source> ::= {CHANnel<n> | EXTernal} for DSO models <source> ::= {CHANnel<n> | DIGital0,..,DIGital15 } for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:IIC[:SOURce]:DATA <source>	:TRIGger:IIC[:SOURce]:DATA?	<source> ::= {CHANnel<n> | EXTernal} for DSO models <source> ::= {CHANnel<n> | DIGital0,..,DIGital15 } for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:IIC:TRIGger:QUALifier <value>	:TRIGger:IIC:TRIGger:QUALifier?	<value> ::= {EQUal | NOTequal | LESSthan | GREaterthan}
:TRIGger:IIC:TRIGger[:TYPE] <type>	:TRIGger:IIC:TRIGger[:TYPE]?	<type> ::= {STARt | STOP | READ7 | READEeprom | WRITe7 | WRITe10 | NACKnowledge | ANACknowledge | R7Data2 | W7Data2 | RESTart}

:TRIGger:IIC Commands

:TRIGger:IIC:PATTern:ADDRess — N

Command Syntax

```
:TRIGger:IIC:PATTern:ADDRess <value>

<value> ::= integer or <string>

<string> ::= "0xnn" where n ::= {0,..,9 | A,..,F}
```

The :TRIGger:IIC:PATTern:ADDRess command sets the address for IIC data.The address can range from 0x00 to 0x7F (7-bit) or 0x3FF (10-bit) hexadecimal. Use the don't care address (-1 or 0xFFFFFFFF) to ignore the address value.

Query Syntax

```
:TRIGger:IIC:PATTern:ADDRess?
```

The :TRIGger:IIC:PATTern:ADDRess? query returns the current address for IIC data.

Return Format

```
<value><NL>
```

```
<value> ::= integer
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:IIC:PATTern:DATA
- :TRIGger:IIC:PATTern:DATa2
- :TRIGger:IIC:TRIGger[:TYPE]

:TRIGger:IIC Commands

:TRIGger:IIC:PATTern:DATA — N

Command Syntax

```
:TRIGger:IIC:PATTern:DATA <value>
```

```
<value> ::= integer or <string>
```

```
<string> ::= "0xnn" where n ::= {0,..,9 | A,..,F}
```

The :TRIGger:IIC:PATTern:DATA command sets IIC data. The data value can range from 0x00 to 0x0FF (hexadecimal). Use the don't care data pattern (-1 or 0xFFFFFFFF) to ignore the data value.

Query Syntax

```
:TRIGger:IIC:PATTern:DATA?
```

The :TRIGger:IIC:PATTern:DATA? query returns the current pattern for IIC data.

Return Format

```
<value><NL>
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:IIC:PATTern:ADDRess
- :TRIGger:IIC:PATTern:DATa2
- :TRIGger:IIC:TRIGger[:TYPE]

:TRIGger:IIC Commands

:TRIGger:IIC:PATTern:DATa2 — N

Command Syntax

```
:TRIGger:IIC:PATTern:DATa2 <value>

<value> ::= integer or <string>

<string> ::= "0xnn" where n ::= {0,..,9 | A,..,F}
```

The :TRIGger:IIC:PATTern:DATa2 command sets IIC data 2. The data value can range from 0x00 to 0x0FF (hexadecimal). Use the don't care data pattern (-1 or 0xFFFFFFFF) to ignore the data value.

Query Syntax

```
:TRIGger:IIC:PATTern:DATa2?
```

The :TRIGger:IIC:PATTern:DATa2? query returns the current pattern for IIC data 2.

Return Format

```
<value><NL>
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:IIC:PATTern:ADDRess
- :TRIGger:IIC:PATTern:DATA
- :TRIGger:IIC:TRIGger[:TYPE]

:TRIGger:IIC Commands

:TRIGger:IIC:SOURce:CLOCk — N

Command Syntax

```
:TRIGger:IIC:[SOURce:]CLOCk <source>

<source> ::= {CHANnel<n> | EXTernal} for the DSO models

<source> ::= {CHANnel<n> | DIGital0,..,DIGital15} for the MSO models

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :TRIGger:IIC:[SOURce:]CLOCk command sets the source for the IIC serial clock (SCL).

Query Syntax

```
:TRIGger:IIC:[SOURce:]CLOCk?
```

The :TRIGger:IIC:[SOURce:]CLOCk? query returns the current source for the IIC serial clock.

Return Format

```
<source><NL>
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:IIC:SOURce:DATA

:TRIGger:IIC Commands

:TRIGger:IIC:SOURce:DATA — N

Command Syntax

```
:TRIGger:IIC:[SOURce:]DATA <source>

<source> ::= {CHANnel<n> | EXTernal} for the DSO models

<source> ::= {CHANnel<n> | DIGital0,..,DIGital15} for the MSO models

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :TRIGger:IIC:[SOURce:]DATA command sets the source for IIC serial data (SDA).

Query Syntax

```
:TRIGger:IIC:[SOURce:]DATA?
```

The :TRIGger:IIC:[SOURce:]DATA? query returns the current source for IIC serial data.

Return Format

```
<source><NL>
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:IIC:SOURce:CLOCk

:TRIGger:IIC Commands

:TRIGger:IIC:TRIGger:QUALifier — N

Command Syntax

```
:TRIGger:IIC:TRIGger:QUALifier <value>

<value> ::= {EQUal | NOTequal | LESSthan | GREaterthan}
```

The :TRIGger:IIC:TRIGger:QUALifier command sets the IIC data qualifier when TRIGger:IIC:TRIGger[:TYPE] is set to READEeprom.

Query Syntax

```
:TRIGger:IIC:TRIGger:QUALifier?
```

The :TRIGger:IIC:TRIGger:QUALifier? query returns the current IIC data qualifier value.

Return Format

```
<value><NL>

<value> ::= {EQUal | NOTequal | LESSthan | GREaterthan}
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:MODE
- :TRIGger:IIC:TRIGger[:TYPE]

:TRIGger:IIC Commands

:TRIGger:IIC:TRIGger[:TYPE] — N

Command Syntax

```
:TRIGger:IIC:TRIGger[:TYPE] <value>

<value> ::= {STARt | STOP | READ7 | READEeprom | WRITe7 | WRITe10
| NACKnowledge | ANACknowledge | R7Data2 | W7Data2 | RESTart}
```

The :TRIGger:IIC:TRIGger[:TYPE] command sets the IIC trigger type:

- STARt — Start condition.
- STOP — Stop condition.
- READ7 — 7-bit address frame containing (Start:Address7:Read:Ack:Data). The value READ is also accepted for READ7.
- R7Data2 — 7-bit address frame containing (Start:Address7:Read:Ack:Data:Ack:Data2).
- READEeprom — EEPROM data read.
- WRITe7 — 7-bit address frame containing (Start:Address7:Write:Ack:Data). The value WRITe is also accepted for WRITe7.
- W7Data2 — 7-bit address frame containing (Start:Address7:Write:Ack:Data:Ack:Data2).

- WRITe10 — 10-bit address frame containing (Start:Address byte1:Write:Ack:Address byte 2:Data).
- NACKnowledge — Missing acknowledge.
- ANACknowledge — Address with no acknowledge.
- RESTart — Another start condition occurs before a stop condition.

NOTE The short form of READ7 (READ7), READEeprom (READE), WRITe7 (WRIT7), and WRITe10 (WRIT10) do not follow the defined Long Form to Short Form Truncation Rules.

Query Syntax

```
:TRIGger:IIC:TRIGger[:TYPE]?
```

The :TRIGger:IIC:TRIGger[:TYPE]? query returns the current IIC trigger type value.

Return Format

```
<value><NL>

<value> ::= {STAR | STOP | READ7 | READE | WRIT7 | WRIT10 | NACK | ANAC
| R7D2 | W7D2 | REST}
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:MODE
- :TRIGger:IIC:PATTern:ADDRess
- :TRIGger:IIC:PATTern:DATA
- :TRIGger:IIC:PATTern:DATa2
- :TRIGger:IIC:TRIGger:QUALifier

:TRIGger Commands

:TRIGger:LIN Commands

Command	Query	Options and Query Returns
:TRIGger:LIN:ID <value>	:TRIGger:LIN:ID?	<value> ::= 7-bit integer in decimal, <nondecimal>, or <string> from 0-63 or 0x00-0x3f (with Option AMS) <nondecimal> ::= #Hnn where n ::= {0,..,9 | A,..,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary <string> ::= "0xnn" where n ::= {0,..,9 | A,..,F} for

		hexadecimal
:TRIGger:LIN:SAMPlepoint <value>	:TRIGger:LIN:SAMPlepoint?	<value> ::= {60 | 62.5 | 68 | 70 | 75 | 80 | 87.5} in NR3 format
:TRIGger:LIN:SIGNal:BAUDrate <baudrate>	:TRIGger:LIN:SIGNal:BAUDrate?	<baudrate> ::= {2400 | 9600 | 19200}
:TRIGger:LIN:SOURce <source>	:TRIGger:LIN:SOURce?	<source> ::= {CHANnel<n> | EXTernal} for DSO models <source> ::= {CHANnel<n> | DIGital0,..,DIGital15} for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:LIN:STANdard <std>	:TRIGger:LIN:STANdard?	<std> ::= {LIN13 | LIN20}
:TRIGger:LIN:SYNCbreak <value>	:TRIGger:LIN:SYNCbreak?	<value> ::= integer = {11 | 12 | 13}
:TRIGger:LIN:TRIGger <condition>	:TRIGger:LIN:TRIGger?	<condition> ::= {SYNCbreak} (without Option AMS) <condition> ::= {SYNCbreak | ID} (with Option AMS)

:TRIGger:LIN Commands

:TRIGger:LIN:ID — N

Command Syntax

```
:TRIGger:LIN:ID <value>

<value> ::= 7-bit integer in decimal, <nondecimal>, or <string> from 0-63 or 0x00-0x3f

<nondecimal> ::= #Hnn where n ::= {0,..,9 | A,..,F} for hexadecimal

<nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary

<string> ::= "0xnn" where n ::= {0,..,9 | A,..,F} for hexadecimal
```

The :TRIGger:LIN:ID command defines the LIN identifier searched for in each CAN message when the LIN trigger mode is set to frame ID.

NOTE This command is only valid when the automotive CAN and LIN serial decode option (Option AMS) has been licensed.

Query Syntax

```
:TRIGger:LIN:ID?
```

The :TRIGger:LIN:ID? query returns the current LIN identifier setting.

Return Format

```
<value><NL>
```

```
<value> ::= integer in decimal
```

Errors

- Hardware missing

See Also

- Introduction to :TRIGger Commands
- :TRIGger:MODE
- :TRIGger:LIN:TRIGger
- :TRIGger:LIN:SIGNal:DEFinition
- :TRIGger:LIN:SOURce

:TRIGger:LIN Commands

:TRIGger:LIN:SAMPlepoint — N

Command Syntax

```
:TRIGger:LIN:SAMPlepoint <value>
```

```
<value><NL>
```

```
<value> ::= {60 | 62.5 | 68 | 70 | 75 | 80 | 87.5} in NR3 format
```

The :TRIGger:LIN:SAMPlepoint command sets the point during the bit time where the bit level is sampled to determine whether the bit is dominant or recessive. The sample point represents the percentage of time between the beginning of the bit time to the end of the bit time.

NOTE The sample point values are not limited by the baud rate.

Query Syntax

```
:TRIGger:LIN:SAMPlepoint?
```

The :TRIGger:LIN:SAMPlepoint? query returns the current LIN sample point setting.

Return Format

```
<value><NL>
```

```
<value> ::= {60 | 62.5 | 68 | 70 | 75 | 80 | 87.5} in NR3 format
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:MODE
- :TRIGger:LIN:TRIGger

:TRIGger:LIN Commands

:TRIGger:LIN:SIGNal:BAUDrate — N

Command Syntax

```
:TRIGger:LIN:SIGNal:BAUDrate <baudrate>

<baudrate> ::= integer in NR1 format

<baudrate> ::= {2400 | 9600 | 19200}
```

The :TRIGger:LIN:SIGNal:BAUDrate command sets the standard baud rate of the LIN signal at 2400 b/s, 9600 b/s, or 19200 b/s. If a non-standard baud rate is sent, the baud rate will be set to the next highest standard rate.

Query Syntax

```
:TRIGger:LIN:SIGNal:BAUDrate?
```

The :TRIGger:LIN:SIGNal:BAUDrate? query returns the current LIN baud rate setting.

Return Format

```
<baudrate><NL>

<baudrate> ::= integer = {2400 | 9600 | 19200}
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:MODE
- :TRIGger:LIN:TRIGger
- :TRIGger:LIN:SIGNal:DEFinition
- :TRIGger:LIN:SOURce

:TRIGger:LIN Commands

:TRIGger:LIN:SOURce — N

Command Syntax

```
:TRIGger:LIN:SOURce <source>

<source> ::= {CHANnel<n> | EXTernal} for the DSO models

<source> ::= {CHANnel<n> | DIGital0,..,DIGital15} for the MSO models

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :TRIGger:LIN:SOURce command sets the source for the LIN signal.

Query Syntax

```
:TRIGger:LIN:SOURce?
```

The :TRIGger:LIN:SOURce? query returns the current source for the LIN signal.

Return Format

```
<source><NL>
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:MODE
- :TRIGger:LIN:TRIGger
- :TRIGger:LIN:SIGNal:DEFinition

:TRIGger:LIN Commands

:TRIGger:LIN:STANdard — N

Command Syntax

```
:TRIGger:LIN:STANdard <std>

<std> ::= {LIN13 | LIN20}
```

The :TRIGger:LIN:STANdard command sets the LIN standard in effect for triggering and decoding to be LIN1.3 or LIN2.0.

Query Syntax

```
:TRIGger:LIN:STANdard?
```

The :TRIGger:LIN:STANdard? query returns the current LIN standard setting.

Return Format

```
<std><NL>

<std> ::= {LIN13 | LIN20}
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:MODE
- :TRIGger:LIN:SIGNal:DEFinition
- :TRIGger:LIN:SOURce

:TRIGger:LIN Commands

:TRIGger:LIN:SYNCbreak — N

Command Syntax

```
:TRIGger:LIN:SYNCbreak <value>

<value> ::= integer = {11 | 12 | 13}
```

The :TRIGger:LIN:SYNCbreak command sets the length of the LIN sync break to be greater than or equal to 11,12, or 13 clock lengths. The sync break is the idle period in the bus activity at the beginning of each packet that distinguishes one information packet from the previous one.

Query Syntax

```
:TRIGger:LIN:SYNCbreak?
```

The :TRIGger:LIN:STANdard? query returns the current LIN sync break setting.

Return Format

```
<value><NL>

<value> ::= {11 | 12 | 13}
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:MODE
- :TRIGger:LIN:SIGNal:DEFinition
- :TRIGger:LIN:SOURce

:TRIGger:LIN Commands

:TRIGger:LIN:TRIGger — N

Command Syntax

```
:TRIGger:LIN:TRIGger <condition>

<condition> ::= {SYNCbreak | ID}
```

The :TRIGger:LIN:TRIGger command sets the LIN trigger on condition to be Sync Break (SYNCbreak) or Frame Id (ID).

NOTE The ID option is available when the automotive CAN and LIN serial decode option (Option AMS) has been licensed.

Query Syntax

```
:TRIGger:LIN:TRIGger?
```

The :TRIGger:LIN:TRIGger? query returns the current LIN trigger value.

Return Format

```
<condition><NL>

<condition> ::= {SYNC | ID}
```

Errors

- Hardware missing

See Also

- Introduction to :TRIGger Commands
- :TRIGger:MODE
- :TRIGger:LIN:SIGNal:DEFinition
- :TRIGger:LIN:SOURce

:TRIGger Commands

:TRIGger:SEQuence Commands

Command	Query	Options and Query Returns
:TRIGger:SEQuence:COUNt <count>	:TRIGger:SEQuence:COUNt?	<count> ::= integer in NR1 format
:TRIGger:SEQuence:EDGE{1|2} <source>, <slope>	:TRIGger:SEQuence:EDGE {1|2}?	<source> ::= {CHANnel<n> | EXTernal} for the DSO models <source> ::= {CHANnel<n> |

		DIGital0,..,DIGital15} for the MSO models <slope> ::= {POSitive | NEGative} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= query returns "NONE" if edge source is disabled
:TRIGger:SEQuence:FIND <value>	:TRIGger:SEQuence:FIND?	<value> ::= {PATTern1,ENTered | PATTern1,EXITed | EDGE1 | PATTern1,AND,EDGE1}
:TRIGger:SEQuence:PATTern {1|2} <value>, <mask>	:TRIGger:SEQuence:PATTern {1|2}?	<value> ::= integer or <string> <mask> ::= integer or <string> <string> ::= "0xnnnnnn" n ::= {0,..,9 | A,..,F}
:TRIGger:SEQuence:RESet <value>	:TRIGger:SEQuence:RESet?	<value> ::= {NONE | PATTern1,ENTered | PATTern1,EXITed | EDGE1 | PATTern1,AND,EDGE1 | PATTern2,ENTered | PATTern2,EXITed | EDGE2 | TIMer} Values used in find and trigger stages not available. EDGE2 not available if EDGE2,COUNt used in trigger stage.
:TRIGger:SEQuence:TIMer <time_value>	:TRIGger:SEQuence:TIMer?	<time_value> ::= time from 100 ns to 10 seconds in NR3 format
:TRIGger:SEQuence:TRIGger <value>	:TRIGger:SEQuence:TRIGger?	<value> ::= {PATTern2,ENTered | PATTern2,EXITed | EDGE2 | PATTern2,AND,EDGE2 | EDGE2,COUNt | EDGE2,COUNt,NREFind}

:TRIGger:SEQuence Commands

:TRIGger:SEQuence:COUNt — N

Command Syntax

```
:TRIGger:SEQuence:COUNt <count>

<count> ::= integer in NR1 format
```

The :TRIGger:SEQuence:COUNt command sets the sequencer edge counter resource. The edge counter is used in the trigger stage to determine the number of edges that must be found before the sequencer generates a trigger.

Query Syntax

```
:TRIGger:SEQuence:COUNt?
```

The :TRIGger:SEQuence:COUNt? query returns the current sequencer edge counter setting.

Return Format

```
<count><NL>
```

```
<count> ::= integer in NR1 format
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:SEQuence:TRIGger
- :TRIGger:SEQuence:EDGE

:TRIGger:SEQuence Commands

:TRIGger:SEQuence:EDGE — N

Command Syntax

```
:TRIGger:SEQuence:EDGE{1 | 2} <source>, <slope>
```

```
<source> ::= {CHANnel<n> | EXTernal} for the DSO models
```

```
<source> ::= {CHANnel<n> | DIGital0,..,DIGital15} for the MSO models
```

```
<slope> ::= {POSitive | NEGative}
```

```
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
```

```
<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :TRIGger:SEQuence:EDGE<n> command defines the specified sequencer edge resource according to the specified <source> and <slope>. To disable an edge resource, set its <source> to NONE. In this case, <slope> has no meaning.

Query Syntax

```
:TRIGger:SEQuence:EDGE{1 | 2}?
```

The :TRIGger:SEQuence:EDGE<n>? query returns the specified sequencer edge resource setting. If the edge resource is disabled, the returned <source> value is NONE. In this case, the <slope> is undefined.

Return Format

```
<source>, <slope><NL>
```

See Also

- Introduction to :TRIGger Commands

- :TRIGger:SEQuence:FIND
- :TRIGger:SEQuence:TRIGger
- :TRIGger:SEQuence:RESet
- :TRIGger:SEQuence:COUNt

:TRIGger:SEQuence Commands

:TRIGger:SEQuence:FIND — N

Command Syntax

```
:TRIGger:SEQuence:FIND <value>

<value> ::= {PATTern1,ENTered | PATTern1,EXITed | EDGE1 | PATTern1,AND,EDGE1}
```

The :TRIGger:SEQuence:FIND command specifies the find stage of a sequence trigger. This command accepts three program data parameters; you can use NONE to fill out the parameter list (for example,"EDGE1,NONE,NONE").

PATTern1 is specified with the :TRIGger:SEQuence:PATTern command. EDGE1 is specified with the :TRIGger:SEQuence:EDGE command.

Query Syntax

```
:TRIGger:SEQuence:FIND?
```

The :TRIGger:SEQuence:FIND? query returns the find stage specification for a sequence trigger. NONE is returned for unused parameters.

Return Format

```
<find_value><NL>

<find_value> ::= {PATT1,ENT,NONE | PATT1,EXIT,NONE | EDGE1,NONE,NONE | PATT1,AND,EDGE1}
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:SEQuence:PATTern
- :TRIGger:SEQuence:EDGE
- :TRIGger:SEQuence:TRIGger
- :TRIGger:SEQuence:RESet

:TRIGger:SEQuence Commands

:TRIGger:SEQuence:PATTern — N

Command Syntax

```
:TRIGger:SEQuence:PATTern{1 | 2} <value>,<mask>

<value> ::= integer or <string>

<mask> ::= integer or <string>

<string> ::= "0xnnnnnn" where n ::= {0,..,9 | A,..,F}
```

The :TRIGger:SEQuence:PATTern<n> command defines the specified sequence pattern resource according to the value and the mask. For both <value> and <mask>, each bit corresponds to a possible trigger channel. The bit assignments vary by instrument:

Oscilloscope Models	Value and Mask Bit Assignments
4 analog + 16 digital channels (mixed-signal)	Bits 0 through 15 - digital channels 0 through 15. Bits 16 through 19 - analog channels 1 through 4.
2 analog + 16 digital channels (mixed-signal)	Bits 0 through 15 - digital channels 0 through 15. Bits 16 and 17 - analog channels 1 and 2.
4 analog channels only	Bits 0 through 3 - analog channels 1 through 4. Bit 4 - external trigger.
2 analog channels only	Bits 0 and 1 - analog channels 1 and 2. Bit 4 - external trigger.

Set a <value> bit to "0" to set the pattern for the corresponding channel to low. Set a <value> bit to "1" to set the pattern to high.

Set a <mask> bit to "0" to ignore the data for the corresponding channel. Only channels with a "1" set on the appropriate mask bit are used.

Query Syntax

```
:TRIGger:SEQuence:PATTern{1 | 2}?
```

The :TRIGger:SEQuence:PATTern<n>? query returns the current settings of the specified pattern resource.

Return Format

```
<value>, <mask><NL>
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:SEQuence:FIND
- :TRIGger:SEQuence:TRIGger
- :TRIGger:SEQuence:RESet

:TRIGger:SEQuence Commands

:TRIGger:SEQuence:RESet — N

Command Syntax

```
:TRIGger:SEQuence:RESet <value>

<value> ::= {NONE | PATTern1,ENTered | PATTern1,EXITed | EDGE1 | PATTern1,AND,EDGE1 |
PATTern2,ENTered | PATTern2,EXITed | EDGE2 | TIMer}

Values used in find and trigger stages are not available.  EDGE2 is not available
if EDGE2,COUNt is used in trigger stage.
```

The :TRIGger:SEQuence:RESet command specifies the reset stage of a sequence trigger. In multi-level trigger specifications, you may find a pattern, then search for another in sequence, but reset the entire search to the beginning if another condition occurs. This command accepts three program data parameters; you can use NONE to fill out the parameter list (for example, "EDGE1,NONE,NONE").

PATTern1 and PATTern2 are specified with the :TRIGger:SEQuence:PATTern command. EDGE1 and EDGE2 are specified with the :TRIGger:SEQuence:EDGE command. TIMer is specified with the :TRIGger:SEQuence:TIMer command.

Query Syntax

```
:TRIGger:SEQuence:RESet?
```

The :TRIGger:SEQuence:RESet? query returns the reset stage specification for a sequence trigger. NONE is returned for unused parameters.

Return Format

```
<reset_value><NL>

<reset_value> ::= {NONE,NONE,NONE | PATT1,ENT,NONE | PATT1,EXIT,NONE | EDGE1,NONE,NONE |
PATT1,AND,EDGE1 | PATT2,ENT,NONE | PATT2,EXIT,NONE | EDGE2,NONE,NONE | TIM,NONE,NONE}
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:SEQuence:PATTern
- :TRIGger:SEQuence:EDGE
- :TRIGger:SEQuence:TIMer
- :TRIGger:SEQuence:FIND
- :TRIGger:SEQuence:TRIGger

:TRIGger:SEQuence Commands

:TRIGger:SEQuence:TIMer — N

Command Syntax

```
:TRIGger:SEQuence:TIMer <time_value>
```

```
<time_value> ::= time in seconds in NR1 format
```

The :TRIGger:SEQuence:TIMer command sets the sequencer timer resource in seconds from 100 ns to 10 s. The timer is used in the reset stage to determine how long to wait for the trigger to occur before restarting.

Query Syntax

```
:TRIGger:SEQuence:TIMer?
```

The :TRIGger:SEQuence:TIMer? query returns current sequencer timer setting.

Return Format

```
<time value><NL>
```

```
<time_value> ::= time in seconds in NR1 format
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:SEQuence:RESet

:TRIGger:SEQuence Commands

:TRIGger:SEQuence:TRIGger — N

Command Syntax

```
:TRIGger:SEQuence:TRIGger <value>
```

```
<value> ::={PATTern2,ENTered | PATTern2,EXITed | EDGE2 |
PATTern2,AND,EDGE2 | EDGE2,COUNt | EDGE2,COUNt,NREFind}
```

The :TRIGger:SEQuence:TRIGger command specifies the trigger stage of a sequence trigger. The sequence commands set various search terms. After all of these are found in sequence, the trigger condition itself is searched for. This command accepts three program data parameters; you can use NONE to fill out the parameter list (for example, "EDGE2,NONE,NONE").

PATTern2 is specified with the :TRIGger:SEQuence:PATTern command. EDGE2 is specified with the :TRIGger:SEQuence:EDGE command. COUNt is specified with the :TRIGger:SEQuence:COUNt command.

Query Syntax

```
:TRIGger:SEQuence:TRIGger?
```

The :TRIGger:SEQuence:TRIGger? query returns the trigger stage specification for a sequence trigger. NONE is returned for unused parameters.

Return Format

```
<trigger_value><NL>

<trigger_value> ::= {PATT2,ENT,NONE | PATT2,EXIT,NONE | EDGE2,NONE,NONE
| PATT2,AND,EDGE2 | EDGE2,COUN,NONE | EDGE2,COUN,NREF}
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:SEQuence:PATTern
- :TRIGger:SEQuence:EDGE
- :TRIGger:SEQuence:COUNt
- :TRIGger:SEQuence:FIND
- :TRIGger:SEQuence:RESet
- :TRIGger:SEQuence:RESet

:TRIGger Commands

:TRIGger:SPI Commands

Command	Query	Options and Query Returns
:TRIGger:SPI:CLOCk:SLOPe <slope>	:TRIGger:SPI:CLOCk:SLOPe?	<slope> ::= {NEGative | POSitive}
:TRIGger:SPI:CLOCk:TIMeout <time_value>	:TRIGger:SPI:CLOCk:TIMeout?	<time_value> ::= time in seconds in NR1 format
:TRIGger:SPI:FRAMing <value>	:TRIGger:SPI:FRAMing?	<value> ::= {CHIPselect | NOTChipselect | TIMeout}
:TRIGger:SPI:PATTern:DATA <value>, <mask>	:TRIGger:SPI:PATTern:DATA?	<value> ::= integer or <string> <mask> ::= integer or <string> <string> ::= "0xnnnnnn" where n ::= {0,..,9 | A,..,F}
:TRIGger:SPI:PATTern:WIDTh <width>	:TRIGger:SPI:PATTern:WIDTh?	<width> ::= integer from 4 to 32 in NR1 format
:TRIGger:SPI:SOURce:CLOCk <source>	:TRIGger:SPI:SOURce:CLOCk?	<value> ::= {CHANnel<n> | EXTernal} for the DSO models <value> ::= {CHANnel<n> | DIGital0,..,DIGital15} for the MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:SPI:SOURce:DATA <source>	:TRIGger:SPI:SOURce:DATA?	<value> ::= {CHANnel<n> | EXTernal} for the DSO models <value> ::= {CHANnel<n> | DIGital0,..,DIGital15} for the MSO models

		`<n> ::= 1-2 or 1-4 in NR1 format`

`:TRIGger:SPI:SOURce:FRAMe <source>`	`:TRIGger:SPI:SOURce:FRAMe?`	`<value> ::= {CHANnel<n> | EXTernal} for the DSO models`
		`<value> ::= {CHANnel<n> | DIGital0,..,DIGital15} for the MSO models`
		`<n> ::= 1-2 or 1-4 in NR1 format`

:TRIGger:SPI Commands

:TRIGger:SPI:CLOCk:SLOPe — N

Command Syntax

```
:TRIGger:SPI:CLOCk:SLOPe <slope>
```

```
<slope> ::= {NEGative | POSitive}
```

The :TRIGger:SPI:CLOCk:SLOPe command specifies the rising edge (POSitive) or falling edge (NEGative) of the SPI clock source that will clock in the data.

Query Syntax

```
:TRIGger:SPI:CLOCk:SLOPe?
```

The :TRIGger:SPI:CLOCk:SLOPe? query returns the current SPI clock source slope.

Return Format

```
<slope><NL>
```

```
<slope> ::= {NEG | POS}
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:SPI:CLOCk:TIMeout
- :TRIGger:SPI:SOURce:CLOCk

:TRIGger:SPI Commands

:TRIGger:SPI:CLOCk:TIMeout — N

Command Syntax

```
:TRIGger:SPI:CLOCk:TIMeout <time_value>
```

```
<time_value> ::= time in seconds in NR1 format
```

The :TRIGger:SPI:CLOCk:TIMeout command sets the SPI signal clock timeout resource in seconds from 500 ns to 10 s when the :TRIGger:SPI:FRAMing command is set to TIMeout. The timer is used to frame a signal by a clock timeout.

Query Syntax

```
:TRIGger:SPI:CLOCk:TIMeout?
```

The :TRIGger:SPI:CLOCk:TIMeout? query returns current SPI clock timeout setting.

Return Format

```
<time value><NL>

<time_value> ::= time in seconds in NR1 format
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:SPI:CLOCk:SLOPe
- :TRIGger:SPI:SOURce:CLOCk
- :TRIGger:SPI:FRAMing

:TRIGger:SPI Commands

:TRIGger:SPI:FRAMing — N

Command Syntax

```
:TRIGger:SPI:FRAMing <value>

<value> ::= {CHIPselect | NOTChipselect | TIMeout}
```

The :TRIGger:SPI:FRAMing command sets the SPI trigger framing value. If TIMeout is selected, the timeout value is set by the :TRIGger:SPI:CLOCk:TIMeout command.

Query Syntax

```
:TRIGger:SPI:FRAMing?
```

The :TRIGger:SPI:FRAMing? query returns the current SPI framing value.

Return Format

```
<value><NL>

<value> ::= {CHIPselect | NOTChipselect | TIMeout}
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:MODE
- :TRIGger:SPI:CLOCk:TIMeout
- :TRIGger:SPI:SOURce:FRAMe

:TRIGger:SPI Commands

:TRIGger:SPI:PATTern:DATA — N

Command Syntax

```
:TRIGger:SPI:PATTern:DATA <value>,<mask>

<value> ::= integer or <string>

<mask> ::= integer or <string>

<string> ::= "0xnnnnnn" where n ::= {0,..,9 | A,..,F}
```

The :TRIGger:SPI:PATTern:DATA command defines the SPI data pattern resource according to the value and the mask. This pattern, along with the data width, control the data pattern searched for in the data stream.

Set a <value> bit to "0" to set the corresponding bit in the data pattern to low. Set a <value> bit to "1" to set the bit to high.

Set a <mask> bit to "0" to ignore that bit in the data stream. Only bits with a "1" set on the mask are used.

Query Syntax

```
:TRIGger:SPI:PATTern:DATA?
```

The :TRIGger:SPI:PATTern:DATA? query returns the current settings of the specified SPI data pattern resource.

Return Format

```
<value>, <mask><NL>
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:SPI:PATTern:WIDTh
- :TRIGger:SPI:SOURce:DATA

:TRIGger:SPI Commands

:TRIGger:SPI:PATTern:WIDTh — N

Command Syntax

```
:TRIGger:SPI:PATTern:WIDTh <width>
```

```
<width> ::= integer from 4 to 32 in NR1 format
```

The :TRIGger:SPI:PATTern:WIDTh command sets the width of the SPI data pattern anywhere from 4 bits to 32 bits.

Query Syntax

```
:TRIGger:SPI:PATTern:WIDTh?
```

The :TRIGger:SPI:PATTern:WIDTh? query returns the current SPI data pattern width setting.

Return Format

```
<width><NL>
```

```
<width> ::= integer from 4 to 32 in NR1 format
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:SPI:PATTern:DATA
- :TRIGger:SPI:SOURce:DATA

:TRIGger:SPI Commands

:TRIGger:SPI:SOURce:CLOCk — N

Command Syntax

```
:TRIGger:SPI:SOURce:CLOCk <source>
```

```
<source> ::= {CHANnel<n> | EXTernal} for the DSO models
```

```
<source> ::= {CHANnel<n> | DIGital0,..,DIGital15} for the MSO models
```

```
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
```

```
<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :TRIGger:SPI:SOURce:CLOCk command sets the source for the SPI serial clock.

Query Syntax

```
:TRIGger:SPI:SOURce:CLOCk?
```

The :TRIGger:SPI:SOURce:CLOCk? query returns the current source for the SPI serial clock.

Return Format

```
<source><NL>
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:SPI:CLOCk:SLOPe
- :TRIGger:SPI:CLOCk:TIMeout
- :TRIGger:SPI:SOURce:FRAMe
- :TRIGger:SPI:SOURce:DATA

:TRIGger:SPI Commands

:TRIGger:SPI:SOURce:DATA — N

Command Syntax

```
:TRIGger:SPI:SOURce:DATA <source>

<source> ::= {CHANnel<n> | EXTernal} for the DSO models

<source> ::= {CHANnel<n> | DIGital0,..,DIGital15} for the MSO models

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :TRIGger:SPI:SOURce:DATA command sets the source for the SPI serial data.

Query Syntax

```
:TRIGger:SPI:SOURce:DATA?
```

The :TRIGger:SPI:SOURce:DATA? query returns the current source for the SPI serial data.

Return Format

```
<source><NL>
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:SPI:SOURce:CLOCk
- :TRIGger:SPI:SOURce:FRAMe
- :TRIGger:SPI:PATTern:DATA
- :TRIGger:SPI:PATTern:WIDTh

:TRIGger:SPI Commands

:TRIGger:SPI:SOURce:FRAMe — N

Command Syntax

```
:TRIGger:SPI:SOURce:FRAMe <source>

<source> ::= {CHANnel<n> | EXTernal} for the DSO models

<source> ::= {CHANnel<n> | DIGital0,..,DIGital15} for the MSO models

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :TRIGger:SPI:SOURce:FRAMe command sets the frame source when :TRIGger:SPI:FRAMing is set to CHIPselect or NOTChipselect.

Query Syntax

```
:TRIGger:SPI:SOURce:FRAMe?
```

The :TRIGger:SPI:SOURce:FRAMe? query returns the current frame source for the SPI serial frame.

Return Format

```
<source><NL>
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:SPI:SOURce:CLOCk
- :TRIGger:SPI:SOURce:DATA
- :TRIGger:SPI:FRAMing

:TRIGger Commands

:TRIGger:TV Commands

Command	Query	Options and Query Returns
:TRIGger:TV:LINE <line number>	:TRIGger:TV:LINE?	<line number> ::= integer in NR1 format
:TRIGger:TV:MODE <tv mode>	:TRIGger:TV:MODE?	<tv mode> ::= {FIEld1 | FIEld2 | AFIelds | ALINes | LINE | VERTical | LFIeld1 | LFIeld2 | LALTernate | LVERtical}
:TRIGger:TV:POLarity <polarity>	:TRIGger:TV:POLarity?	<polarity> ::= {POSitive | NEGative}

:TRIGger:TV:SOURce <source>	:TRIGger:TV:SOURce?	<source> ::= {CHANnel<n>} <n> ::= 1-2 or 1-4 integer in NR1 format
:TRIGger:TV:STANdard <standard>	:TRIGger:TV:STANdard?	<standard> ::= {GENeric | NTSC | PALM | PAL | SECam | {P480L60HZ | P480} | {P720L60HZ | P720} | {P1080L24HZ | P1080} | P1080L25HZ | {I1080L50HZ | I1080} | I1080L60HZ}

:TRIGger:TV Commands

:TRIGger:TV:LINE — N

Command Syntax

```
:TRIGger:TV:LINE <line_number>

<line_number> ::= integer in NR1 format
```

The :TRIGger:TV:LINE command allows triggering on a specific line of video. The line number limits vary with the standard and mode, as shown in the following table.

TV Trigger Line Number Limits:

TV Standard	Mode				
	LINE	LFIeld1	LFIeld2	LALTernate	VERTical
NTSC		1 to 263	1 to 262	1 to 262	
PAL		1 to 313	314 to 625	1 to 312	
PAL-M		1 to 263	264 to 525	1 to 262	
SECAM		1 to 313	314 to 625	1 to 312	
GENERIC		1 to 1024	1 to 1024		1 to 1024
P480L60HZ	1 to 525				
P720L60HZ	1 to 750				
P1080L24HZ	1 to 1125				
P1080L25HZ	1 to 1125				
I1080L50HZ	1 to 1125				
I1080L60HZ	1 to 1125				

Query Syntax

```
:TRIGger:TV:LINE?
```

The :TRIGger:TV:LINE? query returns the current TV trigger line number setting.

Return Format

```
<line_number><NL>
```

```
<line_number>::= integer in NR1 format
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:TV:STANdard
- :TRIGger:TV:MODE

:TRIGger:TV Commands

:TRIGger:TV:MODE — N

Command Syntax

```
:TRIGger:TV:MODE <mode>
```

```
<mode> ::= {FIEld1 | FIEld2 | AFIelds | ALINes | LINE | VERTical | LFIeld1 | LFIeld2 |
LALTernate | LVERtical}
```

The :TRIGger:TV:MODE command selects the TV trigger mode and field. The LVERtical parameter is only available when ::TRIGger:TV:STANdard is GENeric. The LALTernate parameter is not available when :TRIGger:TV:STANdard is GENeric.

Old forms for <mode> are accepted:

<mode>	Old Forms Accepted
FIEld1	F1
FIEld2	F2
AFIeld	ALLFields, ALLFLDS
ALINes	ALLLines
LFIeld1	LINEF1, LINEFIELD1
LFIeld2	LINEF2, LINEFIELD2
LALTernate	LINEAlt
LVERtical	LINEVert

Query Syntax

```
:TRIGger:TV:MODE?
```

The :TRIGger:TV:MODE? query returns the TV trigger mode.

Return Format

```
<value><NL>
```

```
<value> ::= {FIE1 | FIE2 | AFI | ALIN | LINE | VERT | LFI1 | LFI2 | LALT | LVER}
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:TV:STANdard
- :TRIGger:MODE

:TRIGger:TV Commands

:TRIGger:TV:POLarity — N

Command Syntax

```
:TRIGger:TV:POLarity <polarity>

<polarity> ::= {POSitive | NEGative}
```

The :TRIGger:TV:POLarity command sets the polarity for the TV trigger.

Query Syntax

```
:TRIGger:TV:POLarity?
```

The :TRIGger:TV:POLarity? query returns the TV trigger polarity.

Return Format

```
<polarity><NL>

<polarity> ::= {POS | NEG}
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:MODE
- :TRIGger:TV:SOURce

:TRIGger:TV Commands

:TRIGger:TV:SOURce — N

Command Syntax

```
:TRIGger:TV:SOURce <source>

<source> ::= {CHANnel<n>}
```

```
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
```

```
<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :TRIGger:TV:SOURce command selects the channel used to produce the trigger.

Query Syntax

```
:TRIGger:TV:SOURce?
```

The :TRIGger:TV:SOURce? query returns the current TV trigger source.

Return Format

```
<source><NL>
```

```
<source> ::= {CHAN<n>}
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:MODE
- :TRIGger:TV:POLarity

Example Code

- Example Code

:TRIGger:TV Commands

:TRIGger:TV:STANdard — N

Command Syntax

```
:TRIGger:TV:STANdard <standard>
```

```
<standard> ::= {GENeric | NTSC | PALM | PAL | SECam | {P480L60HZ | P480} |
{P720L60HZ | P720} | {P1080L24HZ | P1080} | P1080L25HZ | {I1080L50HZ | I1080} |
I1080L60HZ}
```

The :TRIGger:TV:STANdard command selects the video standard. GENeric mode is non-interlaced.

Query Syntax

```
:TRIGger:TV:STANdard?
```

The :TRIGger:TV:STANdard? query returns the current TV trigger standard setting.

Return Format

```
<standard><NL>
```

```
<standard> ::= {GEN | NTSC | PALM | PAL | SEC | P480L60HZ | P760L60HZ | P1080L24HZ |
P1080L25HZ | I1080L50HZ | I1080L60HZ}
```

:TRIGger Commands

:TRIGger:USB Commands

Command	Query	Options and Query Returns
:TRIGger:USB:SOURce:DMINus <source>	:TRIGger:USB:SOURce:DMINus?	<source> ::= {CHANnel<n> | EXTernal} for the DSO models <source> ::= {CHANnel<n> | DIGital0,..,DIGital15} for the MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:USB:SOURce:DPLus <source>	:TRIGger:USB:SOURce:DPLus?	<source> ::= {CHANnel<n> | EXTernal} for the DSO models <source> ::= {CHANnel<n> | DIGital0,..,DIGital15} for the MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:USB:SPEed <value>	:TRIGger:USB:SPEed?	<value> ::= {LOW | FULL}
:TRIGger:USB:TRIGger <value>	:TRIGger:USB:TRIGger?	<value> ::= {SOP | EOP | ENTersuspend | EXITsuspend | RESet}

:TRIGger:USB Commands

:TRIGger:USB:SOURce:DMINus — N

Command Syntax

```
:TRIGger:USB:SOURce:DMINus <source>

<source> ::= {CHANnel<n> | EXTernal} for the DSO models

<source> ::= {CHANnel<n> | DIGital0,..,DIGital15} for the MSO models

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :TRIGger:USB:SOURce:DMINus command sets the source for the USB D- signal.

Query Syntax

```
:TRIGger:USB:SOURce:DMINus?
```

The :TRIGger:USB:SOURce:DMINus? query returns the current source for the USB D- signal.

Return Format

```
<source><NL>
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:MODE
- :TRIGger:USB:SOURce:DPLus
- :TRIGger:USB:TRIGger

:TRIGger:USB Commands

:TRIGger:USB:SOURce:DPLus — N

Command Syntax

```
:TRIGger:USB:SOURce:DPLus <source>

<source> ::= {CHANnel<n> | EXTernal} for the DSO models

<source> ::= {CHANnel<n> | DIGital0,..,DIGital15} for the MSO models

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :TRIGger:USB:SOURce:DPLus command sets the source for the USB D+ signal.

Query Syntax

```
:TRIGger:USB:SOURce:DPLus?
```

The :TRIGger:USB:SOURce:DPLus? query returns the current source for the USB D+ signal.

Return Format

```
<source><NL>
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:MODE
- :TRIGger:USB:SOURce:DMINus
- :TRIGger:USB:TRIGger

:TRIGger:USB Commands

:TRIGger:USB:SPEed — N

Command Syntax

```
:TRIGger:USB:SPEed <value>
```

```
<value> ::= {LOW | FULL}
```

The :TRIGger:USB:SPEed command sets the expected USB signal speed to be Low Speed (1.5 Mb/s) or Full Speed (12 Mb/s).

Query Syntax

```
:TRIGger:USB:SPEed?
```

The :TRIGger:USB:SPEed? query returns the current speed value for the USB signal.

Return Format

```
<value><NL>
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:MODE
- :TRIGger:USB:SOURce:DMINus
- :TRIGger:USB:SOURce:DPLus
- :TRIGger:USB:TRIGger

:TRIGger:USB Commands

:TRIGger:USB:TRIGger — N

Command Syntax

```
:TRIGger:USB:TRIGger <value>
```

```
<value> ::= {SOP | EOP | ENTersuspend | EXITsuspend | RESet}
```

The :TRIGger:USB:TRIGger command sets where the USB trigger will occur:

- SOP — Start of packet.
- EOP — End of packet.
- ENTersuspend — Enter suspend state.
- EXITsuspend — Exit suspend state.
- RESet — Reset complete.

Query Syntax

```
:TRIGger:USB:TRIGger?
```

The :TRIGger:USB:TRIGger? query returns the current USB trigger value.

Return Format

```
<value><NL>

<value> ::= {SOP | EOP | ENTersuspend | EXITsuspend | RESet}
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:MODE
- :TRIGger:USB:SPEed

Commands by Subsystem

:WAVeform Commands

Provide access to waveform data. See Introduction to :WAVeform Commands.

Command	Query	Options and Query Returns
:WAVeform:BYTeorder <value>	:WAVeform:BYTeorder?	<value> ::= {LSBFirst | MSBFirst}
n/a	:WAVeform:COUNt?	<count> ::= an integer from 1 to 65536 in NR1 format
n/a	:WAVeform:DATA?	<binary block length bytes>, <binary data> For example, to transmit 1000 bytes of data, the syntax would be: #800001000<1000 bytes of data><NL> 8 is the number of digits that follow 00001000 is the number of bytes to be transmitted <1000 bytes of data> is the actual data
:WAVeform:FORMat <value>	:WAVeform:FORMat?	<value> ::= {WORD | BYTE | ASCII}
:WAVeform:POINts <# points>	:WAVeform:POINts?	<# points> ::= {100 | 250 | 500 | 1000 | <points_mode>} if waveform points mode is NORMal <# points> ::= {100 | 250 | 500 | 1000 | 2000 ... 8000000 in 1-2-5 sequence | <points_mode>} if waveform points mode is MAXimum or RAW

		<points_mode> ::= {NORMal | MAXimum | RAW}
:WAVeform:POINts:MODE <points_mode>	:WAVeform:POINts:MODE?	<points_mode> ::= {NORMal | MAXimum | RAW}
n/a	:WAVeform:PREamble?	<preamble_block> ::= <format NR1>, <type NR1>,<points NR1>,<count NR1>, <xincrement NR3>, <xorigin NR3>, <xreference NR1>,<yincrement NR3>, <yorigin NR3>, <yreference NR1> <format> ::= an integer in NR1 format: • 0 for BYTE format • 1 for WORD format • 2 for ASCii format <type> ::= an integer in NR1 format: • 0 for NORMal type • 1 for PEAK detect type • 2 for AVERage type • 3 for HRESolution type <count> ::= Average count, or 1 if PEAK detect type or NORMal; an integer in NR1 format
:WAVeform:SOURce <source>	:WAVeform:SOURce?	<source> ::= {CHANnel<n> | FUNCtion | MATH | SBUS} for DSO models <source> ::= {CHANnel<n> | POD{1 | 2} | BUS {1 | 2} | FUNCtion | MATH | SBUS} for MSO models <n> ::= 1-2 or 1-4 in NR1 format
n/a	:WAVeform:TYPE?	<return_mode> ::= {NORM | PEAK | AVER | HRES}
:WAVeform:UNSigned {{0 | OFF} | {1 | ON}}	:WAVeform:UNSigned?	{0 | 1}
:WAVeform:VIEW <view>	:WAVeform:VIEW?	<view> ::= {MAIN}
n/a	:WAVeform:XINCrement?	<return_value> ::= x-increment in the current preamble in NR3 format
n/a	:WAVeform:XORigin?	<return_value> ::= x-origin value in the current preamble in NR3 format
n/a	:WAVeform:XREFerence?	<return_value> ::= 0 (x-reference value in the current preamble in NR1 format)
n/a	:WAVeform:YINCrement?	<return_value> ::= y-increment value in the current preamble in NR3 format
n/a	:WAVeform:YORigin?	<return_value> ::= y-origin in the current

		preamble in NR3 format
n/a	:WAVeform:YREFerence?	<return_value> ::= y-reference value in the current preamble in NR1 format

Introduction to :WAVeform Commands

The WAVeform subsystem is used to transfer data to a controller from the oscilloscope waveform memories. The queries in this subsystem will only operate when the channel selected by :WAVeform:SOURce is on.

Waveform Data and Preamble. The waveform record is actually contained in two portions: the preamble and waveform data. The waveform record must be read from the oscilloscope by the controller using two separate commands,:WAVeform:DATA and :WAVeform:PREamble. The waveform data is the actual data acquired for each point in the specified source. The preamble contains the information for interpreting the waveform data, which includes the number of points acquired, the format of acquired data, and the type of acquired data. The preamble also contains the X and Y increments, origins, and references for the acquired data, so that word and byte data can be translated to time and voltage values.

Data Acquisition Types. There are three types of waveform acquisitions that can be selected for analog channels with the :ACQuire:TYPE command: NORMal, AVERage, PEAK, and HRESolution. Digital channels are always acquired using NORMal. When the data is acquired using the :DIGitize or:RUN command, the data is placed in the channel buffer of the specified source.

Once you have acquired data with the :DIGitize command, the instrument is stopped. If the instrument is restarted (via GPIB or the front panel), or if any instrument setting is changed, the data acquired with the :DIGitize command may be overwritten.You should first acquire the data with the :DIGitize command, then immediately read the data with the :WAVeform:DATA? query before changing any instrument setup.

A waveform record consists of either all of the acquired points or a subset of the acquired points. The number of points acquired may be queried using :ACQuire:POINts?.

Helpful Hints:

The number of points transferred to the computer is controlled using the :WAVeform:POINts command. If :WAVeform:POINts MAXimum is specified and the instrument is not running (stopped), all of the points that are displayed are transferred. This can be as many as 4,000,000 in some operating modes or as many as 8,000,000 for a digital channel on the mixed signal oscilloscope. Fewer points may be specified to speed data transfers and minimize controller analysis time. The :WAVeform:POINts may be varied even after data on a channel is acquired. However, this decimation may result in lost pulses and transitions. The number of points selected for transfer using :WAVeform:POINts must be an even divisor of 1,000 or be set to MAXimum. :WAVeform:POINts determines the increment between time buckets that will be transferred. If POINts = MAXimum, the data cannot be decimated. For example:

- `:WAVeform:POINts 1000` — returns time buckets 0, 1, 2, 3, 4 ,.., 999.
- `:WAVeform:POINts 500` — returns time buckets 0, 2, 4, 6, 8 ,.., 998.
- `:WAVeform:POINts 250` — returns time buckets 0, 4, 8, 12, 16 ,.., 996.
- `:WAVeform:POINts 100` — returns time buckets 0, 10, 20, 30, 40 ,.., 990.

Analog Channel Data.

NORMal Data

Normal data consists of the last data point (hit) in each time bucket. This data is transmitted over GPIB in a linear fashion starting with time bucket 0 and going through time bucket n - 1, where n is the number returned by the :WAVeform:POINts? query. Only the magnitude values of each data point are transmitted. The first voltage value corresponds to the first time bucket on the left side of the screen and the last value corresponds to the next-to-last time bucket on the right side of the screen. Time buckets without data return 0. The time values for each data point correspond to the position of the data point in the data array. These time values are not transmitted.

AVERage Data

AVERage data consists of the average of the first n hits in a time bucket, where n is the value returned by the :ACQuire:COUNt query. Time buckets that have fewer than n hits return the average of the data they do have. If a time bucket does not have any data in it, it returns 0.

This data is transmitted over the interface linearly, starting with time bucket 0 and proceeding through time bucket n-1, where n is the number returned by the :WAVeform:POINts? query. The first value corresponds to a point at the left side of the screen and the last value corresponds to one point away from the right side of the screen. The maximum number of points that can be returned in average mode is 1000 unless ACQuire:COUNt has been set to 1.

PEAK Data

Peak detect display mode is used to detect glitches for time base settings of 500 us/div and slower. In this mode, the oscilloscope can sample more data than it can store and display. So, when peak detect is turned on, the oscilloscope scans through the extra data, picks up the minimum and maximum for each time bucket, then stores the data in an array. Each time bucket contains two data sample.

The array is transmitted over the interface bus linearly, starting with time bucket 0 proceeding through time bucket n-1, where n is the number returned by the :WAVeform:POINts? query. In each time bucket, two values are transmitted, first the minimum, followed by the maximum. The first pair of values corresponds to the time bucket at the leftmost side of the screen. The last pair of values corresponds to the time bucket at the far right side of the screen. In :ACQuire:TYPE PEAK mode, the value returned by the :WAVeform:XINCrement query should be doubled to find the time difference between the min-max pairs.

HRESolution Data

The high resolution (*smoothing*) mode is used to reduce noise at slower sweep speeds where the digitizer samples faster than needed to fill memory for the displayed time range. This mode is the same as the AVERage mode with :ACQuire:COUNt 1.

Data Conversion

Word or byte data sent from the oscilloscope must be scaled for useful interpretation. The values used to interpret the data are the X and Y references, X and Y origins, and X and Y increments. These values are read from the waveform preamble. Each channel has its own waveform preamble.

In converting a data value to a voltage value, the following formula is used:

voltage = [(data value - yreference) * yincrement] + yorigin

If the :WAVeform:FORMat data format is ASCii, the data values are converted internally and sent as floating point values separated by commas.

In converting a data value to time, the time value of a data point can be determined by the position of the data point. For example, the fourth data point sent with :WAVeform:XORigin = 16 ns, :WAVeform:XREFerence = 0, and :WAVeform:XINCrement = 2 ns, can be calculated using the following formula:

time = [(data point number - xreference) * xincrement] + xorigin

This would result in the following calculation for time bucket 3:

> time = [(3 - 0) * 2 ns] + 16 ns = 22 ns

In :ACQuire:TYPE PEAK mode, because data is acquired in max-min pairs, modify the previous time formula to the following:

> time=[(data pair number - xreference) * xincrement * 2] + xorigin

Data Format for Transfer

There are three formats for transferring waveform data over the interface: BYTE, WORD and ASCii (see :WAVeform:FORMat). BYTE, WORD and ASCii formatted waveform records are transmitted using the arbitrary block program data format specified in IEEE 488.2.

When you use the block data format, the ASCII character string "#8<DD...D>" is sent prior to sending the actual data. The 8 indicates how many Ds follow. The Ds are ASCII numbers that indicate how many data bytes follow.

For example, if 1000 points will be transferred, and the WORD format was specified, the block header "#800001000" would be sent. The 8 indicates that eight length bytes follow, and 00001000 indicates that 1000 binary data bytes follow.

Use the :WAVeform:UNSigned command to control whether data values are sent as unsigned or signed integers. This command can be used to match the instrument's internal data type to the data type used by the programming language. This command has no effect if the data format is ASCii.

Data Format for Transfer - ASCii format

> The ASCii format (see :WAVeform:FORMat) provides access to the waveform data as real Y-axis values without using Y origin, Y reference, and Y increment to convert the binary data. Values are transferred as ASCii digits in floating point format separated by commas. In ASCii format, holes are represented by the value 9.9e+37. The setting of :WAVeform:BYTeorder and :WAVeform:UNSigned have no effect when the format is ASCii.

Data Format for Transfer - WORD format

> WORD format (see :WAVeform:FORMat) provides 16-bit access to the waveform data. In the WORD format, the number of data bytes is twice the number of data points. The number of data points is the value returned by the :WAVeform:POINts? query. If the data intrinsically has less than 16 bits of resolution, the data is left-shifted to provide 16 bits of resolution and the least significant bits are set to 0. Currently, the greatest intrinsic resolution of any data is 12 bits, so at least the lowest 4 bits of data will be 0. If there is a hole in the data, the hole is represented by a 16 bit value equal to 0.
>
> Use :WAVeform:BYTeorder to determine if the least significant byte or most significant byte is to be transferred first. The :BYTeorder command can be used to alter the transmit sequence to match the storage sequence of an integer in the programming language being used.

Data Format for Transfer - BYTE format

> The BYTE format (see :WAVeform:FORMat) allows 8-bit access to the waveform data. If the data intrinsically has more than 8 bits of resolution (averaged data), the data is right-shifted (truncated) to fit into 8 bits. If there is a hole in the data, the hole is represented by a value of 0. The BYTE-formatted data transfers over the GPIB faster than ASCii or WORD-formatted data, because in ASCii format, as many as 13 bytes per point are transferred, in BYTE format one byte per point is transferred, and in WORD format two bytes per point are transferred.
>
> The :WAVeform:BYTeorder command has no effect when the data format is BYTE.

Digital Channel Data (MSO models only). The waveform record for digital channels is similar to that of analog channels. The main difference is that the data points represent either DIGital0,..,7 (POD1), DIGital8,..,15 (POD2), or any grouping of digital channels (BUS1 or BUS2).

For digital channels, :WAVeform:UNSigned must be set to ON.

Digital Channel POD Data Format

Data for digital channels is only available in groups of 8 bits (Pod1 = D0 - D7, Pod2 = D8 - D15). The bytes are organized as:

:WAVeform:SOURce	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
POD1	D7	D6	D5	D4	D3	D2	D1	D0
POD2	D15	D14	D13	D12	D11	D10	D9	D8

If the :WAVeform:FORMat is WORD, every other data byte will be 0. The setting of :WAVeform:BYTeorder controls which byte is 0.

If a digital channel is not displayed, its bit value in the pod data byte is not defined.

Digital Channel BUS Data Format

Digital channel BUS definitions can include any or all of the digital channels. Therefore, data is always returned as 16-bit values. :BUS commands are used to select the digital channels for a bus.

Reporting the Setup. The following is a sample response from the :WAVeform? query. In this case, the query was issued following a *RST command.

```
:WAV:UNS 1;VIEW MAIN;BYT MSBF;FORM WORD;POIN +1000;SOUR CHAN1
```

:WAVeform Commands

:WAVeform:BYTeorder — C

Command Syntax

```
:WAVeform:BYTeorder <value>
```

```
<value> ::= {LSBFirst | MSBFirst}
```

The :WAVeform:BYTeorder command sets the output sequence of the WORD data. The parameter MSBFirst sets the most significant byte to be transmitted first. The parameter LSBFirst sets the least significant byte to be transmitted first. This command affects the transmitting sequence only when :WAVeform:FORMat WORD is selected. The default setting is LSBFirst.

Query Syntax

```
:WAVeform:BYTeorder?
```

The :WAVeform:BYTeorder query returns the current output sequence.

Return Format

```
<value><NL>
```

```
<value> ::= {LSBF | MSBF}
```

See Also

- Introduction to :WAVeform Commands
- :WAVeform:DATA
- :WAVeform:FORMat
- :WAVeform:PREamble

Example Code

- Example Code
- Example Code

:WAVeform Commands

:WAVeform:COUNt — C

Query Syntax

```
:WAVeform:COUNt?
```

The :WAVeform:COUNT? query returns the count used to acquire the current waveform. This may differ from current values if the unit has been stopped and its configuration modified. For all acquisition types except average, this value is 1.

Return Format

```
<count_argument><NL>
```

```
<count_argument> ::= an integer from 1 to 65536 in NR1 format
```

See Also

- Introduction to :WAVeform Commands
- :ACQuire:COUNt
- :ACQuire:TYPE

:WAVeform Commands

:WAVeform:DATA — C

Query Syntax

`:WAVeform:DATA?`

The :WAVeform:DATA query returns the binary block of sampled data points transmitted using the IEEE 488.2 arbitrary block data format. The binary data is formatted according to the settings of the :WAVeform:UNSigned, :WAVeform:BYTeorder, :WAVeform:FORMat, and :WAVeform:SOURce commands. The number of points returned is controlled by the :WAVeform:POINts command.

Return Format

`<binary block data><NL>`

See Also

- Introduction to :WAVeform Commands
- :WAVeform:BYTeorder
- :WAVeform:FORMat
- :WAVeform:POINts
- :WAVeform:PREamble
- :WAVeform:SOURce
- :WAVeform:TYPE

Example Code

```
' QUERY_WAVE_DATA - Outputs waveform data that is stored in a buffer.
myScope.WriteString ":WAV:DATA?"   ' Query the oscilloscope for the waveform data.

' READ_WAVE_DATA - The wave data consists of two parts: the header,
' and the actual waveform data followed by a new line (NL) character.
' The query data has the following format:
'
'    <header><waveform_data><NL>
'
' Where:
'    <header> = #800001000 (This is an example header)
' The "#8" may be stripped off of the header and the remaining
' numbers are the size, in bytes, of the waveform data block.  The
' size can vary depending on the number of points acquired for the
' waveform.  You can then read that number of bytes from the
' oscilloscope and the terminating NL character.
'
Dim lngI As Long
Dim lngDataValue As Long

varQueryResult = myScope.ReadIEEEBlock(BinaryType_UI1)
' Unsigned integer bytes.
For lngI = 0 To UBound(varQueryResult) Step (UBound(varQueryResult) / 20)   ' 20 points.
  If intBytesPerData = 2 Then
    lngDataValue = varQueryResult(lngI) * 256 + varQueryResult(lngI + 1)   ' 16-bit value.
  Else
    lngDataValue = varQueryResult(lngI)   ' 8-bit value.
  End If
  strOutput = strOutput + "Data point " + CStr(lngI / intBytesPerData) + ", " + _
    FormatNumber((lngDataValue - lngYReference) * sngYIncrement + sngYOrigin) + _
    " V, " + _
    FormatNumber(((lngI / intBytesPerData - lngXReference) * sngXIncrement + _
    dblXOrigin) * 1000000) + " us" + vbCrLf
Next lngI
MsgBox "Waveform data:" + vbCrLf + strOutput
```

Example program from the start: VISA COM Example in Visual Basic

:WAVeform Commands

:WAVeform:FORMat — C

Command Syntax

```
:WAVeform:FORMat <value>

<value> ::= {WORD | BYTE | ASCii}
```

The :WAVeform:FORMat command sets the data transmission mode for waveform data points. This command controls how the data is formatted when sent from the oscilloscope.

- ASCii formatted data converts the internal integer data values to real Y-axis values. Values are transferred as ASCii digits in floating point notation, separated by commas.

  ASCII formatted data is transferred ASCii text.
- WORD formatted data transfers 16-bit data as two bytes. The :WAVeform:BYTeorder command can be used to specify whether the upper or lower byte is transmitted first. The default (no command sent) is that the upper byte transmitted first.
- BYTE formatted data is transferred as 8-bit bytes.

When the :WAVeform:SOURce is the serial decode bus (SBUS), ASCii is the only waveform format allowed.

When the :WAVeform:SOURce is one of the digital channel buses (BUS1 or BUS2), ASCii and WORD are the only waveform formats allowed.

Query Syntax

```
:WAVeform:FORMat?
```

The :WAVeform:FORMat query returns the current output format for the transfer of waveform data.

Return Format

```
<value><NL>

<value> ::= {WORD | BYTE | ASC}
```

See Also

- Introduction to :WAVeform Commands
- :WAVeform:BYTeorder
- :WAVeform:DATA
- :WAVeform:PREamble

Example Code

- Example Code

:WAVeform:POINts — C

Command Syntax

```
:WAVeform:POINts <# points>

<# points> ::= {100 | 250 | 500 | 1000 | <points mode>} if waveform points mode is NORMal

<# points> ::= {100 | 250 | 500 | 1000 | 2000 ... 8000000 in 1-2-5 sequence | <points mode>}
if waveform points mode is MAXimum or RAW

<points mode> ::= {NORMal | MAXimum | RAW}
```

NOTE The <points_mode> option is deprecated. Use the :WAVeform:POINts:MODE command instead.

The :WAVeform:POINts command sets the number of waveform points to be transferred with the :WAVeform:DATA? query. This value represents the points contained in the waveform selected with the :WAVeform:SOURce command.

For the analog or digital sources, there are two different records that can be transferred:

- The first is the raw acquisition record. The maximum number of points available in this record is returned by the :ACQuire:POINts? query and may be up to 8,000,000. The raw acquisition record can only be transferred when the oscilloscope is not running and can only be retrieved from the analog or digital sources.
- The second is referred to as the *measurement record* and is a 1000 point (maximum) representation of the raw acquisition record. The measurement record can be retrieved at any time, from any source.

See the :WAVeform:POINts:MODE command for more information on the <points_mode> option.

Only data visible on the display will be returned.

The maximum number of points returned when the waveform source is math or function is 1000.

When the :WAVeform:SOURce is the serial decode bus (SBUS), this command is ignored, and all available serial decode bus data is returned.

Query Syntax

```
:WAVeform:POINts?
```

The :WAVeform:POINts query returns the number of waveform points to be transferred when using the :WAVeform:DATA? query. Setting the points mode will affect what data is transferred (see the :WAVeform:POINts:MODE command for more information).

When the :WAVeform:SOURce is the serial decode bus (SBUS), this query returns the number of messages that were decoded.

Return Format

```
<# points><NL>

<# points> ::= {100 | 250 | 500 | 1000 | <maximum # points>} if waveform points mode is NORMa

<# points> ::= {100 | 250 | 500 | 1000 | 2000 ... 8000000 in 1-2-5 sequence
| <maximum # points>} if waveform points mode is MAXimum or RAW
```

NOTE If a full screen of data is not displayed, the number of points returned will not be 1000 or an even divisor of it.

See Also

- Introduction to :WAVeform Commands
- :ACQuire:POINts
- :WAVeform:DATA
- :WAVeform:VIEW
- :WAVeform:PREamble
- :WAVeform:POINts:MODE

Example Code

```
   ' WAVE_POINTS - Specifies the number of points to be transferred
   ' using the ":WAVEFORM:DATA?" query.
   myScope.WriteString ":WAVEFORM:POINTS 1000"
```

Example program from the start: VISA COM Example in Visual Basic

:WAVeform Commands

:WAVeform:POINts:MODE — N

Command Syntax

```
:WAVeform:POINts:MODE <points_mode>

<points_mode> ::= {NORMal | MAXimum | RAW}
```

The :WAVeform:POINts:MODE command sets the data record to be transferred with the :WAVeform:DATA? query.

For the analog or digital sources, there are two different records that can be transferred:

- The first is the raw acquisition record. The maximum number of points available in this record is returned by the :ACQuire:POINts? query. The raw acquisition record can only be transferred when the oscilloscope is not running and can only be retrieved from the analog or digital sources.
- The second is referred to as the *measurement record* and is a 1000 point (maximum) representation of the raw acquisition record. The measurement record can be retrieved at any time, from any source.

If the <points_mode> is NORMal, the measurement record is retrieved.

If the <points_mode> is RAW, the raw acquisition record is used. Under some conditions, such as when the oscilloscope is running, this data record is unavailable.

If the <points_mode> is MAXimum, whichever record contains the maximum amount of points is used. Usually, this is the raw acquisition record. But, if the raw acquisition record is unavailable (for example, when the oscilloscope is running), or if the reconstruction filter (Sin(x)/x interpolation) is in use, the measurement record may have more data. If data is being retrieved as the oscilloscope is stopped and as the data displayed is changing, the data being retrieved can switch between the measurement and raw acquisition records.

Considerations for MAXimum or RAW data retrieval

- The instrument must be stopped (see the :STOP command or the :DIGitize command in the root subsystem) in order to return more than 1000 points.
- :TIMebase:MODE must be set to MAIN.
- :ACQuire:TYPE must be set to NORMal, AVERage, or HRESolution. If AVERage, :ACQuire:COUNt must be set to 1 in order to return more than 1000 points.
- MAXimum or RAW will allow up to 8,000,000 points to be returned. The number of points returned will vary as the instrument's configuration is changed. Use the :WAVeform:POINts? MAXimum query to determine the maximum number of points that can be retrieved at the current settings.

Query Syntax

```
:WAVeform:POINts:MODE?
```

The :WAVeform:POINts:MODE? query returns the current points mode. Setting the points mode will affect what data is transferred. See the discussion above.

Return Format

```
<points_mode><NL>

<points_mode> ::= {NORMal | MAXimum | RAW}
```

See Also

- Introduction to :WAVeform Commands
- :ACQuire:POINts
- :WAVeform:DATA
- :WAVeform:VIEW
- :WAVeform:PREamble
- :WAVeform:POINts

:WAVeform Commands

:WAVeform:PREamble — C

Query Syntax

```
:WAVeform:PREamble?
```

The :WAVeform:PREamble query requests the preamble information for the selected waveform source. The preamble data contains information concerning the vertical and horizontal scaling of the data of the corresponding channel.

Return Format

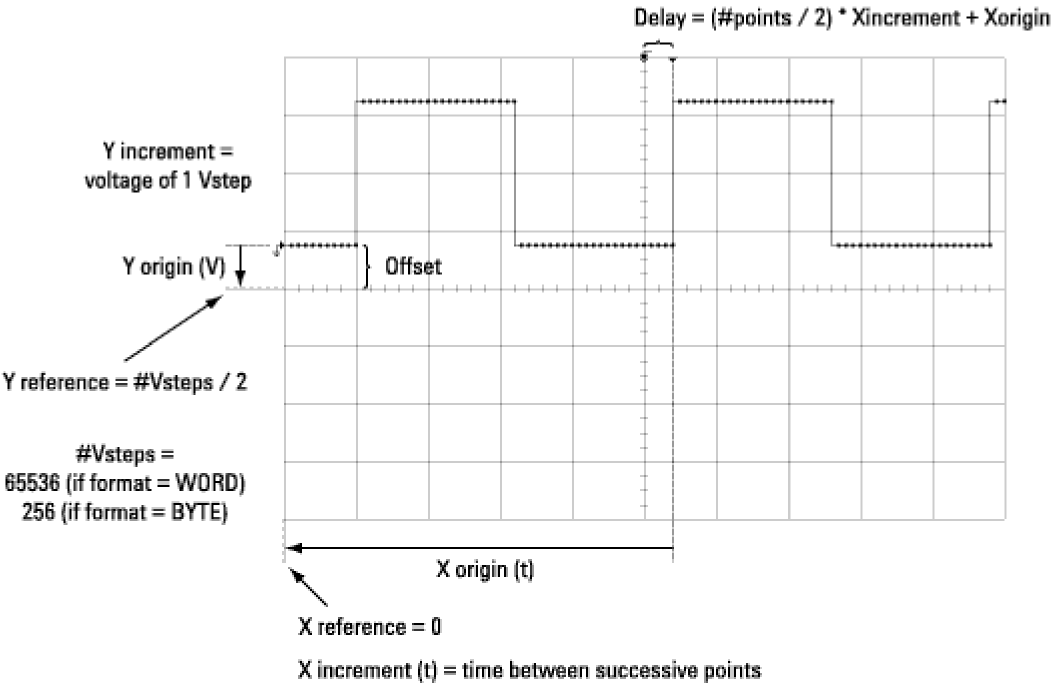
```
<preamble_block><NL>

<preamble_block> ::= <format 16-bit NR1>,
<type 16-bit NR1>,
<points 32-bit NR1>,
<count 32-bit NR1>,
<xincrement 64-bit floating point NR3>,
<xorigin 64-bit floating point NR3>,
<xreference 32-bit NR1>,
<yincrement 32-bit floating point NR3>,
<yorigin 32-bit floating point NR3>,
<yreference 32-bit NR1>

<format> ::= 0 for BYTE format, 1 for WORD format, 2 for ASCii format;
an integer in NR1 format (format set by :WAVeform:FORMat).

<type> ::= 2 for AVERage type, 0 for NORMal type, 1 for PEAK detect type;
an integer in NR1 format (type set by :ACQuire:TYPE).

<count> ::= Average count or 1 if PEAK or NORMal; an integer in NR1 format
(count set by :ACQuire:COUNt).
```


Delay = (#points / 2) * Xincrement + Xorigin
Y increment = voltage of 1 Vstep
Y origin (V)
Offset
Y reference = #Vsteps / 2
#Vsteps = 65536 (if format = WORD) 256 (if format = BYTE)
X origin (t)
X reference = 0
X increment (t) = time between successive points

See Also

- Introduction to :WAVeform Commands
- :ACQuire:COUNt
- :ACQuire:POINts

- :ACQuire:TYPE
- :DIGitize
- :WAVeform:COUNt
- :WAVeform:DATA
- :WAVeform:FORMat
- :WAVeform:POINts
- :WAVeform:TYPE
- :WAVeform:XINCrement
- :WAVeform:XORigin
- :WAVeform:XREFerence
- :WAVeform:YINCrement
- :WAVeform:YORigin
- :WAVeform:YREFerence

Example Code

```
' GET_PREAMBLE - The preamble block contains all of the current
' WAVEFORM settings.  It is returned in the form <preamble_block><NL>
' where <preamble_block> is:
'    FORMAT        : int16 - 0 = BYTE, 1 = WORD, 2 = ASCII.
'    TYPE          : int16 - 0 = NORMAL, 1 = PEAK DETECT, 2 = AVERAGE.
'    POINTS        : int32 - number of data points transferred.
'    COUNT         : int32 - 1 and is always 1.
'    XINCREMENT    : float64 - time difference between data points.
'    XORIGIN       : float64 - always the first data point in memory.
'    XREFERENCE    : int32 - specifies the data point associated with x-origin.
'    YINCREMENT    : float32 - voltage difference between data points.
'    YORIGIN       : float32 - value is the voltage at center screen.
'    YREFERENCE    : int32 - specifies the data point where y-origin occurs.
Dim Preamble()
Dim intFormat As Integer
Dim intType As Integer
Dim lngPoints As Long
Dim lngCount As Long
Dim dblXIncrement As Double
Dim dblXOrigin As Double
Dim lngXReference As Long
Dim sngYIncrement As Single
Dim sngYOrigin As Single
Dim lngYReference As Long
Dim strOutput As String

myScope.WriteString ":WAVEFORM:PREAMBLE?"   ' Query for the preamble.
Preamble() = myScope.ReadList   ' Read preamble information.
intFormat = Preamble(0)
intType = Preamble(1)
lngPoints = Preamble(2)
lngCount = Preamble(3)
dblXIncrement = Preamble(4)
dblXOrigin = Preamble(5)
lngXReference = Preamble(6)
sngYIncrement = Preamble(7)
sngYOrigin = Preamble(8)
lngYReference = Preamble(9)
```

Example program from the start: VISA COM Example in Visual Basic

:WAVeform:SOURce — C

Command Syntax

```
:WAVeform:SOURce <source>

<source> ::= {CHANnel<n> | FUNCtion | MATH | SBUS} for DSO models

<source> ::= {CHANnel<n> | POD{1 | 2} | BUS{1 | 2} | FUNCtion
| MATH | SBUS} for MSO models

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :WAVeform:SOURce command selects the analog channel, function, digital pod, digital bus, or serial decode bus to be used as the source for the :WAVeform commands.

Function capabilities include add, subtract, multiply; integrate, differentiate, and FFT (Fast Fourier Transform) operations.

With MSO oscilloscope models, you can choose a POD or BUS as the waveform source. There are some differences between POD and BUS when formatting and getting data from the oscilloscope:

- When POD1 or POD2 is selected as the waveform source, you can choose the BYTE, WORD, or ASCii formats (see :WAVeform:FORMat).

  When the WORD format is chosen, every other data byte will be 0. The setting of :WAVeform:BYTeorder controls which byte is 0.

  When the ASCii format is chosen, the :WAVeform:DATA? query returns a string with unsigned decimal values separated by commas.
- When BUS1 or BUS2 is selected as the waveform source, you can choose the WORD or ASCii formats (but not BYTE because bus values are always returned as 16-bit values).

  When the ASCii format is chosen, the :WAVeform:DATA? query returns a string with timestamps and hexadecimal bus values, for example: -5.000000000000e-08,0x1938,-4.990000000000e-08,0xff38,...

Query Syntax

```
:WAVeform:SOURce?
```

The :WAVeform:SOURce? query returns the currently selected source for the WAVeform commands.

NOTE MATH is an alias for FUNCtion. The :WAVeform:SOURce? Query returns FUNC if the source is FUNCtion or MATH.

Return Format

```
<source><NL>

<source> ::= {CHAN<n> | FUNC | SBUS} for DSO models
```

```
<source> ::= {CHAN<n> | POD{1 | 2} | BUS{1 | 2} | FUNC | SBUS} for MSO models

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

See Also

- Introduction to :WAVeform Commands
- :DIGitize
- :WAVeform:FORMat
- :WAVeform:DATA
- :WAVeform:PREamble

Example Code

```
' WAVEFORM_DATA - To obtain waveform data, you must specify the
' WAVEFORM parameters for the waveform data prior to sending the
' ":WAVEFORM:DATA?" query.  Once these parameters have been sent,
' the waveform data and the preamble can be read.
'
' WAVE_SOURCE - Selects the channel to be used as the source for
' the waveform commands.
myScope.WriteString ":WAVEFORM:SOURCE CHAN1"

' WAVE_POINTS - Specifies the number of points to be transferred
' using the ":WAVEFORM:DATA?" query.
myScope.WriteString ":WAVEFORM:POINTS 1000"

' WAVE_FORMAT - Sets the data transmission mode for the waveform
' data output.  This command controls whether data is formatted in
' a word or byte format when sent from the oscilloscope.
Dim lngVSteps As Long
Dim intBytesPerData As Integer

myScope.WriteString ":WAVEFORM:FORMAT WORD"   ' Data in range 0 to 65535.
lngVSteps = 65536   intBytesPerData = 2
'myScope.WriteString ":WAVEFORM:FORMAT BYTE"   ' Data in range 0 to 255.
'lngVSteps = 256
'intBytesPerData = 1

' GET_PREAMBLE - The preamble block contains all of the current
' WAVEFORM settings.  It is returned in the form <preamble_block><NL>
' where <preamble_block> is:
'    FORMAT        : int16 - 0 = BYTE, 1 = WORD, 2 = ASCII.
'    TYPE          : int16 - 0 = NORMAL, 1 = PEAK DETECT, 2 = AVERAGE.
'    POINTS        : int32 - number of data points transferred.
'    COUNT         : int32 - 1 and is always 1.
'    XINCREMENT    : float64 - time difference between data points.
'    XORIGIN       : float64 - always the first data point in memory.
'    XREFERENCE    : int32 - specifies the data point associated with x-origin.
'    YINCREMENT    : float32 - voltage difference between data points.
'    YORIGIN       : float32 - value is the voltage at center screen.
'    YREFERENCE    : int32 - specifies the data point where y-origin occurs.
Dim Preamble()
Dim intFormat As Integer
Dim intType As Integer
Dim lngPoints As Long
Dim lngCount As Long
Dim dblXIncrement As Double
Dim dblXOrigin As Double
Dim lngXReference As Long
Dim sngYIncrement As Single
Dim sngYOrigin As Single
Dim lngYReference As Long
Dim strOutput As String

myScope.WriteString ":WAVEFORM:PREAMBLE?"   ' Query for the preamble.
Preamble() = myScope.ReadList   ' Read preamble information.
intFormat = Preamble(0)
intType = Preamble(1)
lngPoints = Preamble(2)
lngCount = Preamble(3)
dblXIncrement = Preamble(4)
dblXOrigin = Preamble(5)
lngXReference = Preamble(6)
sngYIncrement = Preamble(7)
sngYOrigin = Preamble(8)
lngYReference = Preamble(9)
strOutput = ""
'strOutput = strOutput + "Format = " + CStr(intFormat) + vbCrLf
'strOutput = strOutput + "Type = " + CStr(intType) + vbCrLf
'strOutput = strOutput + "Points = " + CStr(lngPoints) + vbCrLf
'strOutput = strOutput + "Count = " + CStr(lngCount) + vbCrLf
'strOutput = strOutput + "X increment = " + FormatNumber(dblXIncrement * 1000000) + _
'            " us" + vbCrLf
'strOutput = strOutput + "X origin = " + FormatNumber(dblXOrigin * 1000000) + _
'            " us" + vbCrLf
'strOutput = strOutput + "X reference = " + CStr(lngXReference) + vbCrLf
'strOutput = strOutput + "Y increment = " + FormatNumber(sngYIncrement * 1000) + _
'           " mV" + vbCrLf
'strOutput = strOutput + "Y origin = " + FormatNumber(sngYOrigin) + " V" + vbCrLf
'strOutput = strOutput + "Y reference = " + CStr(lngYReference) + vbCrLf
strOutput = strOutput + "Volts/Div = " + FormatNumber(lngVSteps * sngYIncrement / 8) + _
            " V" + vbCrLf
strOutput = strOutput + "Offset = " + FormatNumber((lngVSteps / 2 - lngYReference) * _
            sngYIncrement + sngYOrigin) + " V" + vbCrLf
strOutput = strOutput + "Sec/Div = " + FormatNumber(lngPoints * dblXIncrement / 10 * _
            1000000) + " us" + vbCrLf
strOutput = strOutput + "Delay = " + FormatNumber(((lngPoints / 2 - lngXReference) * _
            dblXIncrement + dblXOrigin) * 1000000) + " us" + vbCrLf

' QUERY_WAVE_DATA - Outputs waveform data that is stored in a buffer.
myScope.WriteString ":WAV:DATA?" ' Query the oscilloscope for the waveform data.

' READ_WAVE_DATA - The wave data consists of two parts: the header,
' and the actual waveform data followed by a new line (NL) character.
' The query data has the following format:
'
'    <header><waveform_data><NL>
'
' Where:
'    <header> = #800001000 (This is an example header)
' The "#8" may be stripped off of the header and the remaining
' numbers are the size, in bytes, of the waveform data block.  The
' size can vary depending on the number of points acquired for the
' waveform.  You can then read that number of bytes from the
' oscilloscope and the terminating NL character.
'
Dim lngI As Long
Dim lngDataValue As Long

varQueryResult = myScope.ReadIEEEBlock(BinaryType_UI1)   ' Unsigned integer bytes.
For lngI = 0 To UBound(varQueryResult) Step (UBound(varQueryResult) / 20)   ' 20 points.
  If intBytesPerData = 2 Then
    lngDataValue = varQueryResult(lngI) * 256 + varQueryResult(lngI + 1)   ' 16-bit value.
  Else
```

```
       lngDataValue = varQueryResult(lngI)   ' 8-bit value.
     End If
     strOutput = strOutput + "Data point " + CStr(lngI / intBytesPerData) + ", " + _
       FormatNumber((lngDataValue - lngYReference) * sngYIncrement + sngYOrigin) + _
       " V, " + _
       FormatNumber(((lngI / intBytesPerData - lngXReference) * dblXIncrement + _
       dblXOrigin) * 1000000) + " us" + vbCrLf
   Next lngI
   MsgBox "Waveform data:" + vbCrLf + strOutput
```

Example program from the start: VISA COM Example in Visual Basic

:WAVeform Commands

:WAVeform:TYPE — C

Query Syntax

```
:WAVeform:TYPE?
```

The :WAVeform:TYPE? query returns the acquisition mode associated with the currently selected waveform. The acquisition mode is set by the :ACQuire:TYPE command.

Return Format

```
<mode><NL>

<mode> ::= {NORM | PEAK | AVER | HRES}
```

NOTE If the :WAVeform:SOURce is POD1, POD2, or SBUS, the type is always NORM.

See Also

- Introduction to :WAVeform Commands
- :ACQuire:TYPE
- :WAVeform:DATA
- :WAVeform:PREamble
- :WAVeform:SOURce

:WAVeform Commands

:WAVeform:UNSigned — C

Command Syntax

```
:WAVeform:UNSigned <unsigned>

<unsigned> ::= {{0 | OFF} | {1 | ON}}
```

The :WAVeform:UNSigned command turns unsigned mode on or off for the currently selected waveform. Use the WAVeform:UNSigned command to control whether data values are sent as unsigned or signed integers. This command can be used to match the instrument's internal data type to the data type used by the programming language. This command has no effect if the data format is ASCii.

If :WAVeform:SOURce is set to POD1 or POD2, WAVeform:UNSigned must be set to ON.

Query Syntax

```
:WAVeform:UNSigned?
```

The :WAVeform:UNSigned? query returns the status of unsigned mode for the currently selected waveform.

Return Format

```
<unsigned><NL>

<unsigned> ::= {0 | 1}
```

See Also

- Introduction to :WAVeform Commands

:WAVeform Commands

:WAVeform:VIEW — C

Command Syntax

```
:WAVeform:VIEW <view>

<view> ::= {MAIN}
```

The :WAVeform:VIEW command sets the view setting associated with the currently selected waveform. Currently, the only legal value for the view setting is MAIN.

Query Syntax

```
:WAVeform:VIEW?
```

The :WAVeform:VIEW? query returns the view setting associated with the currently selected waveform.

Return Format

```
<view><NL>

<view> ::= {MAIN}
```

See Also

- Introduction to :WAVeform Commands

- :WAVeform:POINts

:WAVeform Commands

:WAVeform:XINCrement — C

Query Syntax

```
:WAVeform:XINCrement?
```

The :WAVeform:XINCrement? query returns the x-increment value for the currently specified source. This value is the time difference between consecutive data points in seconds.

Return Format

```
<value><NL>

<value> ::= x-increment in the current preamble in 64-bit floating point NR3 format
```

See Also

- Introduction to :WAVeform Commands
- :WAVeform:PREamble

Example Code

- Example Code

:WAVeform Commands

:WAVeform:XORigin — C

Query Syntax

```
:WAVeform:XORigin?
```

The :WAVeform:XORigin? query returns the x-origin value for the currently specified source. XORigin is the X-axis value of the data point specified by the :WAVeform:XREFerence value. In this product, that is always the X-axis value of the first data point (XREFerence = 0).

Return Format

```
<value><NL>

<value> ::= x-origin value in the current preamble in 64-bit floating point NR3 format
```

See Also

- Introduction to :WAVeform Commands

- :WAVeform:PREamble
- :WAVeform:XREFerence

Example Code

- Example Code

:WAVeform Commands

:WAVeform:XREFerence — C

Query Syntax

```
:WAVeform:XREFerence?
```

The :WAVeform:XREFerence? query returns the x-reference value for the currently specified source. This value specifies the index of the data point associated with the x-origin data value. In this product, the x-reference point is the first point displayed and XREFerence is always 0.

Return Format

```
<value><NL>

<value> ::= x-reference value = 0 in 32-bit NR1 format
```

See Also

- Introduction to :WAVeform Commands
- :WAVeform:PREamble
- :WAVeform:XORigin

Example Code

- Example Code

:WAVeform Commands

:WAVeform:YINCrement — C

Query Syntax

```
:WAVeform:YINCrement?
```

The :WAVeform:YINCrement? query returns the y-increment value in volts for the currently specified source. This value is the voltage difference between consecutive data values. The y-increment for digital waveforms is always "1".

Return Format

```
<value><NL>

<value> ::= y-increment value in the current preamble in 32-bit floating point NR3 format
```

See Also

- Introduction to :WAVeform Commands
- :WAVeform:PREamble

Example Code

- Example Code

:WAVeform Commands

:WAVeform:YORigin — C

Query Syntax

```
:WAVeform:YORigin?
```

The :WAVeform:YORigin? query returns the y-origin value for the currently specified source. This value is the Y-axis value of the data value specified by the :WAVeform:YREFerence value. For this product, this is the Y-axis value of the center of the screen.

Return Format

```
<value><NL>

<value> ::= y-origin in the current preamble in 32-bit floating point NR3 format
```

See Also

- Introduction to :WAVeform Commands
- :WAVeform:PREamble
- :WAVeform:YREFerence

Example Code

- Example Code

:WAVeform Commands

:WAVeform:YREFerence — C

Query Syntax

```
:WAVeform:YREFerence?
```

The :WAVeform:YREFerence? query returns the y-reference value for the currently specified source. This value specifies the data point value where the y-origin occurs. In this product, this is the data point value of the center of the screen. It is undefined if the format is ASCii.

Return Format

```
<value><NL>

<value> ::= y-reference value in the current preamble in 32-bit NR1 format
```

See Also

- Introduction to :WAVeform Commands
- :WAVeform:PREamble
- :WAVeform:YORigin

Example Code

- Example Code

Commands A-Z

- :TRIGger:CAN:PATTern:ID
- :TRIGger:CAN:PATTern:ID:MODE
- :TRIGger:CAN:SAMPlepoint
- :TRIGger:CAN:SIGNal:BAUDrate
- :TRIGger:CAN:SIGNal:DEFinition
- :TRIGger:CAN:SOURce
- :TRIGger:CAN:TRIGger
- :TRIGger:DURation:GREaterthan
- :TRIGger:DURation:LESSthan
- :TRIGger:DURation:PATTern
- :TRIGger:DURation:QUALifier
- :TRIGger:DURation:RANGe
- :TRIGger[:EDGE]:COUPling
- :TRIGger[:EDGE]:LEVel
- :TRIGger[:EDGE]:REJect
- :TRIGger[:EDGE]:SLOPe
- :TRIGger[:EDGE]:SOURce
- :TRIGger:GLITch:GREaterthan
- :TRIGger:GLITch:LESSthan
- :TRIGger:GLITch:LEVel
- :TRIGger:GLITch:POLarity
- :TRIGger:GLITch:QUALifier
- :TRIGger:GLITch:RANGe
- :TRIGger:GLITch:SOURce
- :TRIGger:HFReject
- :TRIGger:HOLDoff
- :TRIGger:IIC:PATTern:ADDRess
- :TRIGger:IIC:PATTern:DATA
- :TRIGger:IIC:PATTern:DATa2
- :TRIGger:IIC:SOURce:CLOCk
- :TRIGger:IIC:SOURce:DATA
- :TRIGger:IIC:TRIGger:QUALifier
- :TRIGger:IIC:TRIGger[:TYPE]
- :TRIGger:LIN:SIGNal:BAUDrate
- :TRIGger:LIN:SIGNal:DEFinition
- :TRIGger:LIN:SOURce
- :TRIGger:LIN:TRIGger
- :TRIGger:MODE

Obsolete and Discontinued Commands

Obsolete commands are older forms of commands that are provided to reduce customer rework for existing systems and programs (see O Obsolete Commands).

Obsolete Command	Current Command Equivalent	Behavior Differences
ANALog<n>:BWLimit	:CHANnel<n>:BWLimit	
ANALog<n>:COUPling	:CHANnel<n>:COUPling	
ANALog<n>:INVert	:CHANnel<n>:INVert	
ANALog<n>:LABel	:CHANnel<n>:LABel	
ANALog<n>:OFFSet	:CHANnel<n>:OFFSet	
ANALog<n>:PROBe	:CHANnel<n>:PROBe	
ANALog<n>:PMODe	none	
ANALog<n>:RANGe	:CHANnel<n>:RANGe	
:CHANnel:ACTivity	:ACTivity	
:CHANnel:LABel	:CHANnel<n>:LABel or :DIGital<n>:LABel	use CHANnel<n>:LABel for analog channels and use DIGital<n>:LABel for digital channels
:CHANnel:THReshold	:POD<n>:THReshold or :DIGital<n>:THReshold	
:CHANnel2:SKEW	:CHANnel<n>:PROBe:SKEW	
:CHANnel<n>:INPut	:CHANnel<n>:IMPedance	
:CHANnel<n>:PMODe	none	
:DISPlay:CONNect	:DISPlay:VECTors	
:DISPlay:ORDer	none	
:ERASe	:CDISplay	
:EXTernal:INPut	:EXTernal:IMPedance	
:EXTernal:PMODe	none	
FUNCtion1, FUNCtion2	:FUNCtion Commands	ADD not included
:FUNCtion:VIEW	:FUNCtion:DISPlay	
:HARDcopy:DESTination	:HARDcopy:FILename	
:HARDcopy:DEVice	:HARDcopy:FORMat	PLOTter, THINkjet not supported; TIF, BMP, CSV, SEIko added
:HARDcopy:GRAYscale	:HARDcopy:PALette	
:MEASure:LOWer	:MEASure:DEFine:THResholds	MEASure:DEFine:THResholds can define absolute values or percentage
:MEASure:SCRatch	:MEASure:CLEar	
:MEASure:TDELta	:MARKer:XDELta	
:MEASure:THResholds	:MEASure:DEFine:THResholds	MEASure:DEFine:THResholds can

		define absolute values or percentage
:MEASure:TMAX	:MEASure:XMAX	
:MEASure:TMIN	:MEASure:XMIN	
:MEASure:TSTArt	:MARKer:X1Position	
:MEASure:TSTOp	:MARKer:X2Position	
:MEASure:TVOLt	:MEASure:TVALue	TVALue measures additional values such as db, Vs, etc.
:MEASure:UPPer	:MEASure:DEFine:THResholds	MEASure:DEFine:THResholds can define absolute values or percentage
:MEASure:VDELta	:MARKer:YDELta	
:MEASure:VSTArt	:MARKer:Y1Position	
:MEASure:VSTOp	:MARKer:Y2Position	
:PRINt?	:DISPlay:DATA?	
:TIMebase:DELay	:TIMebase:POSition or :TIMebase:WINDow:POSition	TIMebase:POSition is position value of main time base; TIMebase:WINDow:POSition is position value of delayed time base window.
:TRIGger:CAN:ACKNowledge	none	
:TRIGger:CAN:SIGNal:DEFinition	none	
:TRIGger:LIN:SIGNal:DEFinition	none	
:TRIGger:THReshold	:POD<n>:THReshold or :DIGital<n>:THReshold	
:TRIGger:TV:TVMode	:TRIGger:TV:MODE	

Discontinued Commands

Discontinued commands are commands that were used by previous oscilloscopes, but are not supported by the 6000 Series oscilloscopes. Listed below are the Discontinued commands and the nearest equivalent command available (if any).

Discontinued Command	Current Command Equivalent	Comments
ASTore	:DISPlay:PERSistence INFinite	
CHANnel:MATH	:FUNCtion:OPERation	ADD not included
CHANnel<n>:PROTect	:CHANnel<n>:PROTection	Previous form of this command was used to enable/disable 50Ω protection. The new command resets a tripped protect and the query returns the status of TRIPed or NORMal.
DISPlay:INVerse	none	
DISPlay:COLumn	none	

DISPlay:GRID	none	
DISPLay:LINE	none	
DISPlay:PIXel	none	
DISPlay:POSition	none	
DISPlay:ROW	none	
DISPlay:TEXT	none	
FUNCtion:MOVE	none	
FUNCtion:PEAKs	none	
HARDcopy:ADDRess	none	Only parallel printer port is supported. GPIB printing not supported
MASK	none	All commands discontinued, feature not available
SYSTem:KEY	none	
TEST:ALL	*TST (Self Test)	
TRACE subsystem	none	All commands discontinued, feature not available
TRIGger:ADVanced subsystem		Use new GLITch, PATTern, or TV trigger modes
TRIGger:TV:FIELd	:TRIGger:TV:MODE	
TRIGger:TV:TVHFrej		
TRIGger:TV:VIR	none	
VAUToscale	none	

Discontinued Parameters

Some previous oscilloscope queries returned control setting values of OFF and ON. The 6000 Series oscilloscopes only return the enumerated values 0 (for off) and 1 (for on).

Obsolete and Discontinued Commands

:CHANnel:ACTivity — O

Command Syntax

```
:CHANnel:ACTivity
```

The :CHANnel:ACTivity command clears the cumulative edge variables for the next activity query.

NOTE The :CHANnel:ACTivity command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :ACTivity command for the 6000 Series oscilloscopes.

Query Syntax

```
:CHANnel:ACTivity?
```

The :CHANnel:ACTivity? query returns the active edges since the last clear, and returns the current logic levels.

Return Format

```
<edges>,<levels><NL>

<edges> ::= presence of edges (32-bit integer in NR1 format).

<levels> ::= logical highs or lows (32-bit integer in NR1 format).
```

NOTE A bit equal to zero indicates that no edges were detected at the specified threshold since the last clear on that channel. Edges may have occurred that were not detected because of the threshold setting.

A bit equal to one indicates that edges have been detected at the specified threshold since the last clear on that channel.

Obsolete and Discontinued Commands

:CHANnel:LABel — O

Command Syntax

```
:CHANnel:LABel <source_text><string>

<source_text> ::= {CHANnel1 | CHANnel2 | DIGital0,..,DIGital15}

<string> ::= quoted ASCII string
```

The :CHANnel:LABel command sets the source text to the string that follows. Setting a channel will also result in the name being added to the label list.

NOTE The :CHANnel:LABel command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :CHANnel<n>:LABel or :DIGital<n>:LABel command for the 6000 Series oscilloscopes.

Query Syntax

```
:CHANnel:LABel?
```

The :CHANnel:LABel? query returns the label associated with a particular analog channel.

Return Format

```
<string><NL>

<string> ::= quoted ASCII string
```

Obsolete and Discontinued Commands

:CHANnel:THReshold — O

Command Syntax

```
:CHANnel:THReshold <channel group>, <threshold type> [, <value>]

<channel group> ::= {POD1 | POD2}

<threshold type> ::= {CMOS | ECL | TTL | USERdef}

<value> ::= voltage for USERdef in NR3 format [volt_type]

[volt_type] ::= {V | mV (-3) | uV (-6)}
```

The :CHANnel:THReshold command sets the threshold for a group of channels. The threshold is either set to a predefined value or to a user-defined value. For the predefined value, the voltage parameter is ignored.

NOTE The :CHANnel:THReshold command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :POD<n>:THReshold or :DIGital<n>:THReshold command for the 6000 Series oscilloscopes.

Query Syntax

```
:CHANnel:THReshold? <channel group>
```

The :CHANnel:THReshold? query returns the voltage and threshold text for a specific group of channels.

Return Format

```
<threshold type> [, <value>]<NL>

<threshold type> ::= {CMOS | ECL | TTL | USERdef}

<value> ::= voltage for USERdef (float 32 NR3)
```

NOTE

- CMOS = 2.5V
- TTL = 1.5V
- ECL = -1.3V
- USERdef ::= -6.0V to 6.0V

Obsolete and Discontinued Commands

:CHANnel2:SKEW — O

Command Syntax

```
:CHANnel2:SKEW <skew value>
```

```
<skew value> ::= skew time in NR3 format

<skew value> ::= -100 ns to +100 ns
```

The :CHANnel2:SKEW command sets the skew between channels 1 and 2. The maximum skew is +/-100 ns. You can use the oscilloscope's analog probe skew control to remove cable delay errors between channel 1 and channel 2.

NOTE The :CHANnel2:SKEW command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :CHANnel<n>:PROBe:SKEW command for the 6000 Series oscilloscopes.

NOTE This command is only valid for the two channel oscilloscope models.

Query Syntax

```
:CHANnel2:SKEW?
```

The :CHANnel2:SKEW? query returns the current probe skew setting for the selected channel.

Return Format

```
<skew value><NL>

<skew value> ::= skew value in NR3 format
```

See Also

- Introduction to :CHANnel<n> Commands

Obsolete and Discontinued Commands

:CHANnel<n>:INPut — O

Command Syntax

```
:CHANnel<n>:INPut <impedance>

<impedance> ::= {ONEMeg | FIFTy}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :CHANnel<n>:INPut command selects the input impedance setting for the specified channel. The legal values for this command are ONEMeg (1 MΩ) and FIFTy (50Ω).

NOTE The :CHANnel<n>:INPut command is an obsolete command provided for compatibility to previous oscilloscopes. Use the:CHANnel<n>:IMPedance command for the 6000 Series oscilloscopes.

Query Syntax

```
:CHANnel<n>:INPut?
```

The :CHANnel<n>:INPut? query returns the current input impedance setting for the specified channel.

Return Format

```
<impedance value><NL>

<impedance value> ::= {ONEM | FIFT}
```

Obsolete and Discontinued Commands

:CHANnel<n>:PMODe — O

Command Syntax

```
:CHANnel<n>:PMODe <pmode value>

<pmode value> ::= {AUTo | MANual}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The probe sense mode is controlled internally and cannot be set. If a probe with sense is connected to the specified channel, auto sensing is enabled; otherwise, the mode is manual.

If the PMODe sent matches the oscilloscope's setting, the command will be accepted. Otherwise, a setting conflict error is generated.

NOTE The :CHANnel<n>:PMODe command is an obsolete command provided for compatibility to previous oscilloscopes.

Query Syntax

```
:CHANnel<n>:PMODe?
```

The :CHANnel<n>:PMODe? query returns AUT if an autosense probe is attached and MAN otherwise.

Return Format

```
<pmode value><NL>

<pmode value> ::= {AUT | MAN}
```

Obsolete and Discontinued Commands

:DISPlay:CONNect — O

Command Syntax

```
:DISPlay:CONNect <connect>
```

```
<connect> ::= {{ 1 | ON} | {0 | OFF}}
```

The :DISPlay:CONNect command turns vectors on and off. When vectors are turned on, the oscilloscope displays lines connecting sampled data points. When vectors are turned off, only the sampled data is displayed.

NOTE The :DISPlay:CONNEct command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :DISPlay:VECTors command for the 6000 Series oscilloscopes.

Query Syntax

```
:DISPlay:CONNect?
```

The :DISPlay:CONNect? query returns the current state of the vectors setting.

Return Format

```
<connect><NL>
```

```
<connect> ::= {1 | 0}
```

See Also

- :DISPlay:VECTors

Obsolete and Discontinued Commands

:DISPlay:ORDer — O

Query Syntax

```
:DISPlay:ORDer?
```

The :DISPlay:ORDer? query returns a list of digital channel numbers in screen order, from top to bottom, separated by commas. Busing is displayed as digital channels with no separator. For example, in the following list, the bus consists of digital channels 4 and 5: DIG1, DIG4 DIG5, DIG7.

NOTE The :DISPlay:ORDer command is an obsolete command provided for compatibility to previous oscilloscopes. This command is only available on the MSO models.

Return Format

```
<order><NL>
```

`<order> ::= Unquoted ASCII string`

NOTE A return value is included for each digital channel. A return value of NONE indicates that a channel is turned off.

See Also

- :DIGital<n>:POSition

Example Code

```
' DISP_ORDER - Set the order the channels are displayed on the
' analyzer.  You can enter between 1 and 32 channels at one time.
' If you leave out channels, they will not be displayed.
myScope.WriteString ":DISPLAY:ORDER 0,10"   ' Display ONLY channel 0 and
                                            ' channel 10 in that order.
```

Example program from the start: VISA COM Example in Visual Basic

Obsolete and Discontinued Commands

:ERASe — O

Command Syntax

`:ERASe`

The :ERASe command erases the screen.

NOTE The :ERASe command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :CDISplay command for the 6000 Series oscilloscopes.

Obsolete and Discontinued Commands

:EXTernal:INPut — O

Command Syntax

`:EXTernal:INPut <impedance>`

`<impedance> ::= {ONEMeg | FIFTy}`

The :EXTernal:IMPedance command selects the input impedance setting for the external trigger. The legal values for this command are ONEMeg (1 MΩ) and FIFTy (50Ω).

NOTE The :EXTernal:INPut command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :EXTernal:IMPedance command for the 6000 Series oscilloscopes.

Query Syntax

```
:EXTernal:INPut?
```

The :EXTernal:INPut? query returns the current input impedance setting for the external trigger.

Return Format

```
<impedance value><NL>
```

```
<impedance value> ::= {ONEM | FIFT}
```

See Also

- Introduction to :EXTernal Trigger Commands
- Introduction to :TRIGger Commands
- :CHANnel<n>:IMPedance

Obsolete and Discontinued Commands

:EXTernal:PMODe — O

Command Syntax

```
:EXTernal:PMODe <pmode value>
```

```
<pmode value> ::= {AUTo | MANual}
```

The probe sense mode is controlled internally and cannot be set. If a probe with sense is connected to the specified channel, auto sensing is enabled; otherwise, the mode is manual.

If the pmode sent matches the oscilloscope's setting, the command will be accepted. Otherwise, a setting conflict error is generated.

NOTE The :EXTernal:PMODe command is an obsolete command provided for compatibility to previous oscilloscopes.

Query Syntax

```
:EXTernal:PMODe?
```

The :EXTernal:PMODe? query returns AUT if an autosense probe is attached and MAN otherwise.

Return Format

```
<pmode value><NL>
```

```
<pmode value> ::= {AUT | MAN}
```

Obsolete and Discontinued Commands

:FUNCtion:VIEW — O

Command Syntax

```
:FUNCtion:VIEW <view>

<view> ::= {{1 | ON} | (0 | OFF}}
```

The :FUNCtion:VIEW command turns the selected function on or off. When ON is selected, the function performs as specified using the other FUNCtion commands. When OFF is selected, function is neither calculated nor displayed.

NOTE The :FUNCtion:VIEW command is provided for backward compatibility to previous oscilloscopes. Use the :FUNCtion:DISPlay command for the 6000 Series oscilloscopes.

Query Syntax

```
:FUNCtion:VIEW?
```

The :FUNCtion:VIEW? query returns the current state of the selected function.

Return Format

```
<view><NL>

<view> ::= {1 | 0}
```

Obsolete and Discontinued Commands

:HARDcopy:DESTination — O

Command Syntax

```
:HARDcopy:DESTination <destination>

<destination> ::= {CENTronics | FLOPpy}
```

The :HARDcopy:DESTination command sets the hardcopy destination.

NOTE The :HARDcopy:DESTination command is an obsolete command provided for compatibility to previous oscilloscopes. Use the:HARDcopy:FILename command for the 6000 Series oscilloscopes.

Query Syntax

```
:HARDcopy:DESTination?
```

The :HARDcopy:DESTination? query returns the selected hardcopy destination.

Return Format

```
<destination><NL>

<destination> ::= {CENT | FLOP}
```

See Also

- Introduction to :HARDcopy Commands
- :HARDcopy:FORMat

Obsolete and Discontinued Commands

:HARDcopy:DEVice — O

Command Syntax

```
:HARDcopy:DEVice <device>

<device> ::= {TIFF | GIF | BMP | LASerjet | EPSon | DESKjet | BWDeskjet | SEIKo}
```

The HARDcopy:DEVice command sets the hardcopy device type.

NOTE BWDeskjet option refers to the monochrome Deskjet printer.

NOTE The :HARDcopy:DEVice command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :HARDcopy:FORMat command for the 6000 Series oscilloscopes.

Query Syntax

```
:HARDcopy:DEVice?
```

The :HARDcopy:DEVice? query returns the selected hardcopy device type.

Return Format

```
<device><NL>

<device> ::= {TIFF | GIF | BMP | LAS | EPS | DESK | BWD | SEIK}
```

Obsolete and Discontinued Commands

:HARDcopy:GRAYscale — O

Command Syntax

```
:HARDcopy:GRAYscale <gray>

<gray> ::= {{OFF | 0} | {ON | 1}}
```

The :HARDcopy:GRAYscale command controls whether grayscaling is performed in the hardcopy dump.

NOTE The :HARDcopy:GRAYscale command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :HARDcopy:PALette command for the 6000 Series oscilloscopes. (":HARDcopy:GRAYscale ON" is the same as ":HARDcopy:PALette GRAYscale" and ":HARDcopy:GRAYscale OFF" is the same as ":HARDcopy:PALette COLor".)

Query Syntax

```
:HARDcopy:GRAYscale?
```

The :HARDcopy:GRAYscale? query returns a flag indicating whether grayscaling is performed in the hardcopy dump.

Return Format

```
<gray><NL>

<gray> ::= {0 | 1}
```

See Also

- Introduction to :HARDcopy Commands

:MEASure:LOWer — O

Command Syntax

```
:MEASure:LOWer <voltage>
```

The :MEASure:LOWer command sets the lower measurement threshold value. This value and the UPPer value represent absolute values when the thresholds are ABSolute and percentage when the thresholds are PERCent as defined by the :MEASure:DEFine THResholds command.

NOTE The :MEASure:LOWer command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :MEASure:DEFine THResholds command for the 6000 Series oscilloscopes.

Query Syntax

```
:MEASure:LOWer?
```

The :MEASure:LOWer? query returns the current lower threshold level.

Return Format

```
<voltage><NL>

<voltage> ::= the user-defined lower threshold in volts in NR3 format
```

See Also

- Introduction to :MEASure Commands
- :MEASure:THResholds
- :MEASure:UPPer

Obsolete and Discontinued Commands

:MEASure:SCRatch — O

Command Syntax

```
:MEASure:SCRatch
```

The :MEASure:SCRatch command clears all selected measurements and markers from the screen.

NOTE The :MEASure:SCRatch command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :MEASure:CLEar command for the 6000 Series oscilloscopes.

Obsolete and Discontinued Commands

:MEASure:TDELta — O

Query Syntax

```
:MEASure:TDELta?
```

The :MEASure:TDELta? query returns the time difference between the Tstop marker (X2 cursor) and the Tstart marker (X1 cursor).

Tdelta = Tstop - Tstart

Tstart is the time at the start marker (X1 cursor) and Tstop is the time at the stop marker (X2 cursor). No measurement is made when the :MEASure:TDELta? query is received by the oscilloscope. The delta time value that is output is the current value. This is the same value as the front-panel cursors delta X value.

NOTE The :MEASure:TDELta command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :MARKer:XDELta command for the 6000 Series oscilloscopes.

Return Format

```
<value><NL>
```

```
<value> ::= time difference between start and stop markers in NR3 format
```

See Also

- Introduction to :MARKer Commands
- Introduction to :MEASure Commands
- :MARKer:X1Position
- :MARKer:X2Position
- :MARKer:XDELta
- :MEASure:TSTArt
- :MEASure:TSTOp

Obsolete and Discontinued Commands

:MEASure:THResholds — O

Command Syntax

```
:MEASure:THResholds {T1090 | T2080 | VOLTage}
```

The :MEASure:THResholds command selects the thresholds used when making time measurements.

NOTE The :MEASure:THResholds command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :MEASure:DEFine THResholds command for the 6000 Series oscilloscopes.

Query Syntax

```
:MEASure:THResholds?
```

The :MEASure:THResholds? query returns the current thresholds selected when making time measurements.

Return Format

```
{T1090 | T2080 | VOLTage}<NL>

{T1090} uses the 10% and 90% levels of the selected waveform.

{T2080} uses the 20% and 80% levels of the selected waveform.

{VOLTage} uses the upper and lower voltage thresholds set by the
UPPer and LOWer commands on the selected waveform.
```

See Also

- Introduction to :MEASure Commands
- :MEASure:LOWer
- :MEASure:UPPer

Obsolete and Discontinued Commands

:MEASure:TMAX — O

Command Syntax

```
:MEASure:TMAX [<source>]

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:TMAX command installs a screen measurement and starts an X-at-Max-Y measurement on the selected waveform. If the optional source is specified, the current source is modified.

NOTE The :MEASure:TMAX command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :MEASure:XMAX command for the 6000 Series oscilloscopes.

Query Syntax

```
:MEASure:TMAX? [<source>]
```

The :MEASure:TMAX? query returns the horizontal axis value at which the maximum vertical value occurs on the current source. If the optional source is specified, the current source is modified. If all channels are off, the query returns 9.9E+37.

Return Format

```
<value><NL>

<value> ::= time at maximum in NR3 format
```

See Also

- Introduction to :MEASure Commands
- :MEASure:TMIN
- :MEASure:XMAX
- :MEASure:XMIN

Obsolete and Discontinued Commands

:MEASure:TMIN — O

Command Syntax

```
:MEASure:TMIN [<source>]

<source> ::= {CHANnel<n> | FUNCtion | MATH}
```

```
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
```

```
<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:TMIN command installs a screen measurement and starts an X-at-Min-Y measurement on the selected waveform. If the optional source is specified, the current source is modified.

NOTE The :MEASure:TMIN command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :MEASure:XMIN command for the 6000 Series oscilloscopes.

Query Syntax

```
:MEASure:TMIN? [<source>]
```

The :MEASure:TMIN? query returns the horizontal axis value at which the minimum vertical value occurs on the current source. If the optional source is specified, the current source is modified. If all channels are off, the query returns 9.9E+37.

Return Format

```
<value><NL>
```

```
<value> ::= time at minimum in NR3 format
```

See Also

- Introduction to :MEASure Commands
- :MEASure:TMAX
- :MEASure:XMAX
- :MEASure:XMIN

Obsolete and Discontinued Commands

:MEASure:TSTArt — O

Command Syntax

```
:MEASure:TSTArt <value> [suffix]
```

```
<value> ::= time at the start marker in seconds
```

```
[suffix] ::= {s | ms | us | ns | ps}
```

The :MEASure:TSTArt command moves the start marker (X1 cursor) to the specified time with respect to the trigger time.

NOTE The short form of this command, TSTA, does not follow the defined Long Form to Short Form Truncation Rules. The normal short form "TST" would be the same for both TSTArt and TSTOp, so sending TST for the TSTArt command produces an error.

NOTE The :MEASure:TSTArt command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :MARKer:X1Position command for the 6000 Series oscilloscopes.

Query Syntax

```
:MEASure:TSTArt?
```

The :MEASure:TSTArt? query returns the time at the start marker (X1 cursor).

Return Format

```
<value><NL>

<value> ::= time at the start marker in NR3 format
```

See Also

- Introduction to :MARKer Commands
- Introduction to :MEASure Commands
- :MARKer:X1Position
- :MARKer:X2Position
- :MARKer:XDELta
- :MEASure:TDELta
- :MEASure:TSTOp

Obsolete and Discontinued Commands

:MEASure:TSTOp — O

Command Syntax

```
:MEASure:TSTOp <value> [suffix]

<value> ::= time at the stop marker in seconds

[suffix] ::= {s | ms | us | ns | ps}
```

The :MEASure:TSTOp command moves the stop marker (X2 cursor) to the specified time with respect to the trigger time.

NOTE The short form of this command, TSTO, does not follow the defined Long Form to Short Form Truncation Rules. The normal short form "TST" would be the same for both TSTArt and TSTOp, so sending TST for the TSTOp command produces an error.

NOTE The :MEASure:TSTOp command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :MARKer:X2Position command for the 6000 Series oscilloscopes.

Query Syntax

```
:MEASure:TSTOp?
```

The :MEASure:TSTOp? query returns the time at the stop marker (X2 cursor).

Return Format

```
<value><NL>

<value> ::= time at the stop marker in NR3 format
```

See Also

- Introduction to :MARKer Commands
- Introduction to :MEASure Commands
- :MARKer:X1Position
- :MARKer:X2Position
- :MARKer:XDELta
- :MEASure:TDELta
- :MEASure:TSTArt

Obsolete and Discontinued Commands

:MEASure:TVOLt — O

Query Syntax

```
:MEASure:TVOLt? <value>, [<slope>]<occurrence>[,<source>]

<value> ::= the voltage level that the waveform must cross.

<slope> ::= direction of the waveform.  A rising slope is indicated by a
plus sign (+). A falling edge is indicated by a minus sign (-).

<occurrence> ::= the transition to be reported.  If the occurrence number
is one, the first crossing is reported.  If the number is two, the second
crossing is reported, etc.

<source> ::= {<digital channels> | CHANnel<n> | FUNCtion | MATH}

<digital channels> ::= {DIGital0,..,DIGital15} for the MSO models

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

When the :MEASure:TVOLt? query is sent, the displayed signal is searched for the specified voltage level and transition. The time interval between the trigger event and this defined occurrence is returned as the response to the query.

The specified voltage can be negative or positive. To specify a negative voltage, use a minus sign (-). The sign of the slope selects a rising (+) or falling (-) edge. If no sign is specified for the slope, it is assumed to be the rising edge.

The magnitude of the occurrence defines the occurrence to be reported. For example, +3 returns the time for the third time the waveform crosses the specified voltage level in the positive direction. Once this voltage crossing is found, the oscilloscope reports the time at that crossing in seconds, with the trigger point (time zero) as the reference.

If the specified crossing cannot be found, the oscilloscope reports +9.9E+37. This value is returned if the waveform does not cross the specified voltage, or if the waveform does not cross the specified voltage for the specified number of times in the direction specified.

If the optional source parameter is specified, the current source is modified.

NOTE The :MEASure:TVOLt command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :MEASure:TVALue command for the 6000 Series oscilloscopes.

Return Format

```
<value><NL>
```

```
<value> ::= time in seconds of the specified voltage crossing in NR3 format
```

Obsolete and Discontinued Commands

:MEASure:UPPer —

Command Syntax

```
:MEASure:UPPer <value>
```

The :MEASure:UPPer command sets the upper measurement threshold value. This value and the LOWer value represent absolute values when the thresholds are ABSolute and percentage when the thresholds are PERCent as defined by the :MEASure:DEFine THResholds command.

NOTE The :MEASure:UPPer command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :MEASure:DEFine THResholds command for the 6000 Series oscilloscopes.

Query Syntax

```
:MEASure:UPPer?
```

The :MEASure:UPPer? query returns the current upper threshold level.

Return Format

```
<value><NL>
```

```
<value> ::= the user-defined upper threshold in NR3 format
```

See Also

- Introduction to :MEASure Commands
- :MEASure:LOWer
- :MEASure:THResholds

Obsolete and Discontinued Commands

:MEASure:VDELta — O

Query Syntax

```
:MEASure:VDELta?
```

The :MEASure:VDELta? query returns the voltage difference between vertical marker 1 (Y1 cursor) and vertical marker 2 (Y2 cursor). No measurement is made when the :MEASure:VDELta? query is received by the oscilloscope. The delta value that is returned is the current value. This is the same value as the front-panel cursors delta Y value.

VDELta = value at marker 2 - value at marker 1

NOTE The :MEASure:VDELta command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :MARKer:YDELta command for the 6000 Series oscilloscopes.

Return Format

```
<value><NL>

<value> ::= delta V value in NR1 format
```

See Also

- Introduction to :MARKer Commands
- Introduction to :MEASure Commands
- :MARKer:Y1Position
- :MARKer:Y2Position
- :MARKer:YDELta
- :MEASure:TDELta
- :MEASure:TSTArt

Obsolete and Discontinued Commands

:MEASure:VSTArt — O

Command Syntax

```
:MEASure:VSTArt <vstart_argument>
```

```
<vstart_argument> ::= value for vertical marker 1
```

The :MEASure:VSTArt command moves the vertical marker (Y1 cursor) to the specified value corresponding to the selected source. The source can be selected by the MARKer:X1Y1source command.

NOTE The short form of this command, VSTA, does not follow the defined Long Form to Short Form Truncation Rules. The normal short form, VST, would be the same for both VSTArt and VSTOp, so sending VST for the VSTArt command produces an error.

NOTE The :MEASure:VSTArt command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :MARKer:Y1Position command for the 6000 Series oscilloscopes.

Query Syntax

```
:MEASure:VSTArt?
```

The :MEASure:VSTArt? query returns the current value of the Y1 cursor.

Return Format

```
<value><NL>

<value> ::= voltage at voltage marker 1 in NR3 format
```

See Also

- Introduction to :MARKer Commands
- Introduction to :MEASure Commands
- :MARKer:Y1Position
- :MARKer:Y2Position
- :MARKer:YDELta
- :MARKer:X1Y1source
- :MEASure:SOURce
- :MEASure:TDELta
- :MEASure:TSTArt

Obsolete and Discontinued Commands

:MEASure:VSTOp — O

Command Syntax

```
:MEASure:VSTOp <vstop_argument>

<vstop_argument> ::= value for Y2 cursor
```

The :MEASure:VSTOp command moves the vertical marker 2 (Y2 cursor) to the specified value corresponding to the selected source. The source can be selected by the MARKer:X2Y2source command.

NOTE The short form of this command, VSTO, does not follow the defined Long Form to Short Form Truncation Rules. The normal short form, VST, would be the same for both VSTArt and VSTOp, so sending VST for the VSTOp command produces an error.

NOTE The :MEASure:VSTOp command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :MARKer:Y2Position command for the 6000 Series oscilloscopes.

Query Syntax

```
:MEASure:VSTOp?
```

The :MEASure:VSTOp? query returns the current value of the Y2 cursor.

Return Format

```
<value><NL>

<value> ::= value of the Y2 cursor in NR3 format
```

See Also

- Introduction to :MARKer Commands
- Introduction to :MEASure Commands
- :MARKer:Y1Position
- :MARKer:Y2Position
- :MARKer:YDELta
- :MARKer:X2Y2source
- :MEASure:SOURce
- :MEASure:TDELta
- :MEASure:TSTArt

Obsolete and Discontinued Commands

:PRINt? — O

Query Syntax

```
:PRINt? [<options>]

<options> ::= [<print option>][,..,<print option>]

<print option> ::= {COLor | GRAYscale | BMP8bit | BMP}
```

The :PRINt? query pulls image data back over the bus for storage.

NOTE The :PRINT command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :DISPlay:DATA command for the 6000 Series oscilloscopes.

Print Option	:PRINt command	:PRINt? query	Query Default
COLor	Sets palette=COLor		
GRAYscale	Sets palette=GRAYscale		palette=COLor
PRINter0,1	Causes the USB printer #0,1 to be selected as destination (if connected)	Not used	N/A
BMP8bit	Sets print format to 8-bit BMP	Selects 8-bit BMP formatting for query	N/A
BMP	Sets print format to BMP	Selects BMP formatting for query	N/A
FACTors	Selects outputting of additional settings information for :PRINT	Not used	N/A
NOFactors	Deselects outputting of additional settings information for :PRINT	Not used	N/A

Old Print Option:	Is Now:
HIRes	COLor
LORes	GRAYscale
PARallel	PRINter0
DISK	invalid
PCL	invalid

NOTE The PRINt? query is not a core command.

See Also

- Introduction to Root (:) Commands
- Introduction to :HARDcopy Commands
- :HARDcopy:FORMat
- :HARDcopy:FACTors
- :HARDcopy:GRAYscale
- :DISPlay:DATA

Obsolete and Discontinued Commands

:TIMebase:DELay — O

Command Syntax

```
:TIMebase:DELay <delay_value>
```

```
<delay_value> ::= time in seconds from trigger to the delay reference point
on the screen.
```

```
The valid range for delay settings depends on the time/division setting for
the main time base.
```

The :TIMebase:DELay command sets the main time base delay. This delay is the time between the trigger event and the delay reference point on the screen. The delay reference point is set with the:TIMebase:REFerence command.

NOTE The :TIMebase:DELay command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :TIMebase:POSition command for the 6000 Series oscilloscopes.

Query Syntax

```
:TIMebase:DELay?
```

The :TIMebase:DELay query returns the current delay value.

Return Format

```
<delay_value><NL>
```

```
<delay_value> ::= time from trigger to display reference in seconds in NR3 format.
```

Example Code

```
   ' TIMEBASE_DELAY - Sets the time base delay.  This delay
   ' is the internal time between the trigger event and the
   ' onscreen delay reference point.
   myScope.WriteString ":TIMEBASE:DELAY 0.0"   ' Set time base delay to 0.0.
```

Example program from the start: VISA COM Example in Visual Basic

Obsolete and Discontinued Commands

:TRIGger:CAN:ACKNowledge — O

Command Syntax

```
:TRIGger:CAN:ACKNowledge <value>
```

```
<value> ::= {0 | OFF}
```

This command was used with the N2758A CAN trigger module for 54620/54640 Series mixed-signal oscilloscopes. The 6000 Series oscilloscopes do not support the N2758A CAN trigger module.

Query Syntax

```
:TRIGger:CAN:ACKNowledge?
```

The :TRIGger:CAN:ACKNowledge? query returns the current CAN acknowledge setting.

Return Format

```
<value><NL>

<value> ::= 0
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:MODE
- :TRIGger:CAN:TRIGger

Obsolete and Discontinued Commands

:TRIGger:CAN:SIGNal:DEFinition — O

Command Syntax

```
:TRIGger:CAN:SIGNal:DEFinition <value>

<value> ::= {CANH | CANL | RX | TX | DIFFerential}
```

The :TRIGger:CAN:SIGNal:DEFinition command sets the CAN signal type when :TRIGger:CAN:TRIGger is set to SOF (start of frame). These signals can be set to:

Dominant high signal:

- CANH — the actual CAN_H differential bus signal.

Dominant low signals:

- CANL — the actual CAN_L differential bus signal.
- RX — the Receive signal from the CAN bus transceiver.
- TX — the Transmit signal to the CAN bus transceiver.
- DIFFerential — the CAN differential bus signal connected to an analog source channel using a differential probe.

NOTE With 6000 Series oscilloscope software version 3.50 or greater, this command is available, but the only legal value is DIFF.

Query Syntax

```
:TRIGger:CAN:SIGNal:DEFinition?
```

The :TRIGger:CAN:SIGNal:DEFinition? query returns the current CAN signal type.

Return Format

```
<value><NL>

<value> ::= DIFF
```

See Also

- Introduction to :TRIGger Commands
- :TRIGger:MODE
- :TRIGger:CAN:SIGNal:BAUDrate
- :TRIGger:CAN:SOURce
- :TRIGger:CAN:TRIGger

Obsolete and Discontinued Commands

:TRIGger:LIN:SIGNal:DEFinition — O

Command Syntax

```
:TRIGger:LIN:SIGNal:DEFinition <value>

<value> ::= {LIN | RX | TX}
```

The :TRIGger:LIN:SIGNal:DEFinition command sets the LIN signal type. These signals can be set to:

Dominant low signals:

- LIN — the actual LIN single-end bus signal line.
- RX — the Receive signal from the LIN bus transceiver.
- TX — the Transmit signal to the LIN bus transceiver.

NOTE With 6000 Series oscilloscope software version 3.50 or greater, this command is available, but the only legal value is LIN.

Query Syntax

```
:TRIGger:LIN:SIGNal:DEFinition?
```

The :TRIGger:LIN:SIGNal:DEFinition? query returns the current LIN signal type.

Return Format

```
<value><NL>

<value> ::= LIN
```

See Also

- Introduction to :TRIGger Commands

- :TRIGger:MODE
- :TRIGger:LIN:SIGNal:BAUDrate
- :TRIGger:LIN:SOURce

:TRIGger:THReshold — O

Command Syntax

```
:TRIGger:THReshold <channel group>, <threshold type> [, <value>]

<channel group> ::= {POD1 | POD2}

<threshold type> ::= {CMOS | ECL | TTL | USERdef}

<value>::= voltage for USERdef (floating-point number) [Volt type]

[Volt type] ::= {V | mV | uV}
```

The :TRIGger:THReshold command sets the threshold (trigger level) for a pod of 8 digital channels (either digital channels 0 through 7 or 8 through 15). The threshold can be set to a predefined value or to a user-defined value. For the predefined value, the voltage parameter is not required.

NOTE This command is only available on the MSO models.

NOTE The :TRIGger:THReshold command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :POD<n>:THReshold or :DIGital<n>:THReshold or:TRIGger[:EDGE]:LEVel command for the 6000 Series oscilloscopes.

Query Syntax

```
:TRIGger:THReshold? <channel group>
```

The :TRIGger:THReshold? query returns the voltage and threshold text for analog channel 1 or 2, or POD1 or POD2.

Return Format

```
<threshold type>[, <value>]<NL>

<threshold type> ::= {CMOS | ECL | TTL | USER}

CMOS ::= 2.5V

TTL ::= 1.5V

ECL ::= -1.3V

USERdef ::= range from -8.0V to +8.0V.
```

```
<value> ::= voltage for USERdef (a floating-point number in NR1.
```

Obsolete and Discontinued Commands

:TRIGger:TV:TVMode — O

Command Syntax

```
:TRIGger:TV:TVMode <mode>
```

```
<mode> ::= {FIEld1 | FIEld2 | AFIelds | ALINes | LINE | VERTical | LFIeld1
| LFIeld2 | LALTernate | LVERtical}
```

The :TRIGger:TV:MODE command selects the TV trigger mode and field. The LVERtical parameter is only available when ::TRIGger:TV:STANdard is GENeric. The LALTernate parameter is not available when :TRIGger:TV:STANdard is GENeric.

Old forms for <mode> are accepted:

<mode>	Old Forms Accepted
FIEld1	F1
FIEld2	F2
AFIeld	ALLFields, ALLFLDS
ALINes	ALLLines
LFIeld1	LINEF1, LINEFIELD1
LFIeld2	LINEF2, LINEFIELD2
LALTernate	LINEAlt
LVERtical	LINEVert

NOTE The :TRIGger:TV:TVMode command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :TRIGger:TV:MODE command for the 6000 Series oscilloscopes.

Query Syntax

```
:TRIGger:TV:TVMode?
```

The :TRIGger:TV:TVMode? query returns the TV trigger mode.

Return Format

```
<value><NL>
```

```
<value> ::= {FIE1 | FIE2 | AFI | ALIN | LINE | VERT | LFI1 | LFI2 | LALT | LVER}
```

Syntax Elements

Syntax Elements

Number Format

NR1 specifies integer data.

NR3 specifies exponential data in floating point format (for example, -1.0E-3).

Syntax Elements

<NL> (Line Terminator)

<NL> = new line or linefeed (ASCII decimal 10).

The line terminator, or a leading colon, will send the parser to the "root" of the command tree.

Syntax Elements

[ ] (Optional Syntax Terms)

Items enclosed in square brackets, [ ], are optional.

Syntax Elements

{ } (Braces)

When several items are enclosed by braces, { }, only one of these elements may be selected. Vertical line (|) indicates "or". For example, {ON | OFF} indicates that only ON or OFF may be selected, not both.

Syntax Elements

::= (Defined As)

::= means "defined as".

For example, <A> ::= <B> indicates that <A> can be replaced by <B> in any statement containing <A>.

Syntax Elements

< > (Angle Brackets)

< > Angle brackets enclose words or characters that symbolize a program code parameter or an interface command.

Syntax Elements

... (Ellipsis)

... An ellipsis (trailing dots) indicates that the preceding element may be repeated one or more times.

Syntax Elements

n,..,p (Value Ranges)

n,..,p ::= all integers between n and p inclusive.

Syntax Elements

d (Digits)

d ::= A single ASCII numeric character 0 - 9.

Syntax Elements

Quoted ASCII String

A quoted ASCII string is a string delimited by either double quotes (") or single quotes ('). Some command parameters require a quoted ASCII string. For example, when using the Agilent VISA COM library in Visual Basic, the command:

```
myScope.WriteString ":CHANNEL1:LABEL 'One'"
```

has a quoted ASCII string of:

'One'

In order to read quoted ASCII strings from query return values, some programming languages require special handling or syntax.

Syntax Elements

Definite-Length Block Response Data

Definite-length block response data allows any type of device-dependent data to be transmitted over the system interface as a series of 8-bit binary data bytes. This is particularly useful for sending large quantities of data or 8-bit extended ASCII codes. This syntax is a pound sign (#) followed by a non-zero digit representing the number of digits in the decimal integer. After the non-zero digit is the decimal integer that states the number of 8-bit data bytes being sent. This is followed by the actual data.

For example, for transmitting 1000 bytes of data, the syntax would be

```
#800001000<1000 bytes of data> <NL>
```

8 is the number of digits that follow

00001000 is the number of bytes to be transmitted

<1000 bytes of data> is the actual data

Error Messages

-440 Query UNTERMINATED after indefinite response

-430 Query DEADLOCKED

-420 Query UNTERMINATED

-410 Query INTERRUPTED

-400 Query error

-340 Calibration failed

-330 Self-test failed

-321 Out of memory

-320 Storage fault

-315 Configuration memory lost

-314 Save/recall memory lost

-313 Calibration memory lost

-311

	Memory error
-310	**System error**
-300	**Device specific error**
-278	**Macro header not found**
-277	**Macro redefinition not allowed**
-276	**Macro recursion error**
-273	**Illegal macro label**
-272	**Macro execution error**
-258	**Media protected**
-257	**File name error**
-256	**File name not found**
-255	**Directory full**
-254	**Media full**
-253	**Corrupt media**
-252	**Missing media**
-251	**Missing mass storage**
-250	**Mass storage error**
-241	**Hardware missing** This message can occur when a feature is unavailable or unlicensed. For example, serial bus decode commands (which require a four-channel oscilloscope) are unavailable on two-channel oscilloscopes, and some serial bus decode commands are only available on four-channel oscilloscopes when the AMS (automotive serial decode) or LSS (low-speed serial decode) options are licensed.
-240	**Hardware error**
-231	**Data questionable**

-230	Data corrupt or stale
-224	Illegal parameter value
-223	Too much data
-222	Data out of range
-221	Settings conflict
-220	Parameter error
-200	Execution error
-183	Invalid inside macro definition
-181	Invalid outside macro definition
-178	Expression data not allowed
-171	Invalid expression
-170	Expression error
-168	Block data not allowed
-161	Invalid block data
-158	String data not allowed
-151	Invalid string data
-150	String data error
-148	Character data not allowed
-138	Suffix not allowed
-134	Suffix too long
-131	Invalid suffix
-128	Numeric data not allowed

-124	Too many digits
-123	Exponent too large
-121	Invalid character in number
-120	Numeric data error
-114	Header suffix out of range
-113	Undefined header
-112	Program mnemonic too long
-109	Missing parameter
-108	Parameter not allowed
-105	GET not allowed
-104	Data type error
-103	Invalid separator
-102	Syntax error
-101	Invalid character
-100	Command error
10	Software Fault Occurred
100	File Exists
101	End-Of-File Found
102	Read Error
103	Write Error
104	Illegal Operation
105	Print Canceled

106 Print Initialization Failed

107 Invalid Trace File

108 Compression Error

109 No Data For Operation

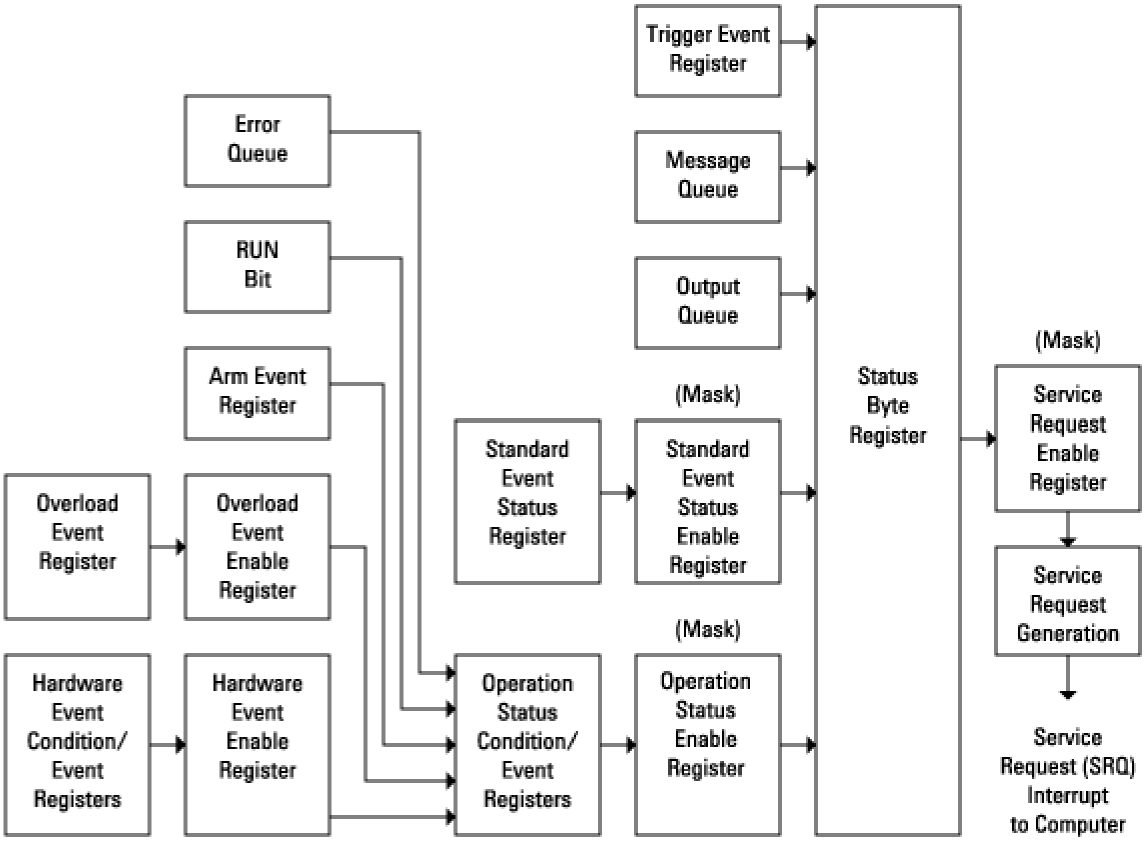
112 Unknown File Type

113 Directory Not Supported

Status Reporting

IEEE 488.2 defines data structures, commands, and common bit definitions for status reporting (for example, the Status Byte Register and the Standard Event Status Register). There are also instrument-defined structures and bits (for example, the Operation Status Event Register and the Overload Event Register).

An overview of the oscilloscope’s status reporting structure is shown in the following block diagram. The status reporting structure allows monitoring specified events in the oscilloscope. The ability to monitor and report these events allows determination of such things as the status of an operation, the availability and reliability of the measured data, and more.


Trigger Event Register
Message Queue
Output Queue
Error Queue
RUN Bit
Arm Event Register
Overload Event Register
Overload Event Enable Register
Hardware Event Condition/ Event Registers
Hardware Event Enable Register
Standard Event Status Register
(Mask)
Standard Event Status Enable Register
(Mask)
Operation Status Condition/ Event Registers
Operation Status Enable Register
Status Byte Register
(Mask)
Service Request Enable Register
Service Request Generation
Service Request (SRQ) Interrupt to Computer

- To monitor an event, first clear the event; then, enable the event. All of the events are cleared when you initialize the instrument.
- To allow a service request (SRQ) interrupt to an external controller, enable at least one bit in the Status Byte Register (by setting, or unmasking, the bit in the Service Request Enable register).

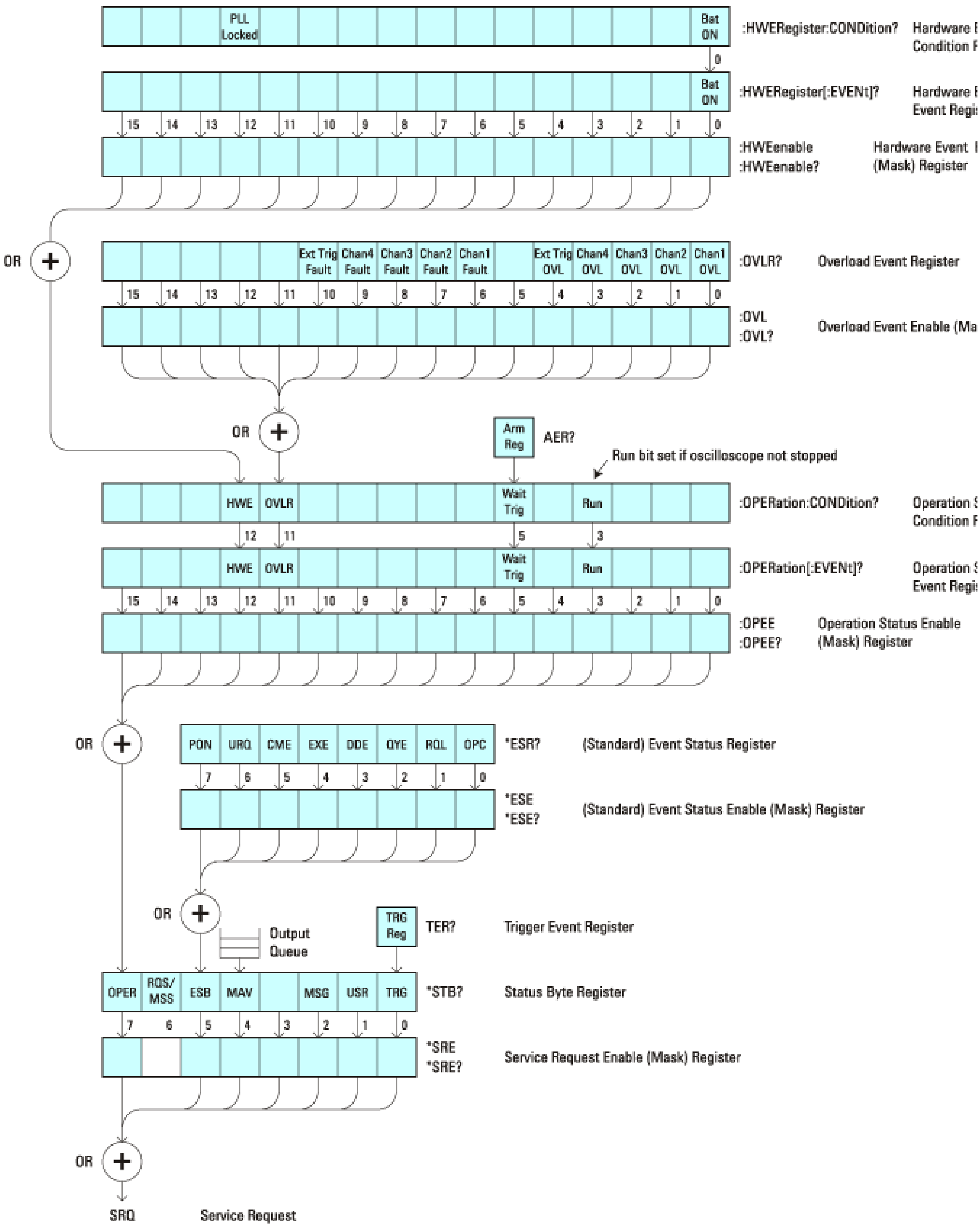
The Status Byte Register, the Standard Event Status Register group, and the Output Queue are defined as the Standard Status Data Structure Model in IEEE 488.2-1987.

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 with 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 you send *CLS immediately after a program message terminator, the output queue is also cleared.

Status Reporting

Status Reporting Data Structures

The following figure shows how the status register bits are masked and logically OR’ed to generate service requests (SRQ) on particular events.



The status register bits are described in more detail in the following tables:

- Status Byte Register (STB)
- Standard Event Status Register (ESR)
- Operation Status Condition Register
- Operation Status Event Register
- Overload Event Register (OVLR)
- Hardware Event Condition Register
- Hardware Event Event Register

The status registers picture above shows how the different status reporting data structures work together. To make it possible for any of the Standard Event Status Register bits to generate a summary bit, the bits must be enabled. These bits are enabled by using the *ESE common command to set the corresponding bit in the Standard Event Status Enable Register.

To generate a service request (SRQ) interrupt to an external controller, at least one bit in the Status Byte Register must be enabled. These bits are enabled by using the *SRE common command to set the corresponding bit in the Service Request Enable Register. These enabled bits can then set RQS and MSS (bit 6) in the Status Byte Register.

Status Reporting

Status Byte Register (STB)

The Status Byte Register is the summary-level register in the status reporting structure. It contains summary bits that monitor activity in the other status registers and queues. The Status Byte Register is a live register. That is, its summary bits are set and cleared by the presence and absence of a summary bit from other event registers or queues.

If the Status Byte Register is to be used with the Service Request Enable Register to set bit 6 (RQS/MSS) and to generate an SRQ, at least one of the summary bits must be enabled, then set. Also, event bits in all other status registers must be specifically enabled to generate the summary bit that sets the associated summary bit in the Status Byte Register.

The Status Byte Register can be read using either the *STB? Common Command or the GPIB serial poll command. Both commands return the decimal-weighted sum of all set bits in the register. The difference between the two methods is that the serial poll command reads bit 6 as the Request Service (RQS) bit and clears the bit which clears the SRQ interrupt. The *STB? command reads bit 6 as the Master Summary Status (MSS) and does not clear the bit or have any affect on the SRQ interrupt. The value returned is the total bit weights of all of the bits that are set at the present time.

The use of bit 6 can be confusing. This bit was defined to cover all possible computer interfaces, including a computer that could not do a serial poll. The important point to remember is that, if you are using an SRQ interrupt to an external computer, the serial poll command clears bit 6. Clearing bit 6 allows the oscilloscope to generate another SRQ interrupt when another enabled event occurs.

No other bits in the Status Byte Register are cleared by either the *STB? query or the serial poll, except the Message Available bit (bit 4). If there are no other messages in the Output Queue, bit 4 (MAV) can be cleared as a result of reading the response to the *STB? command.

If bit 4 (weight = 16) and bit 5 (weight = 32) are set, the program prints the sum of the two weights. Since these bits were not enabled to generate an SRQ, bit 6 (weight = 64) is not set.

The following example uses the *STB? query to read the contents of the oscilloscope’s Status Byte Register.

```
myScope.WriteString "*STB?"
varQueryResult = myScope.ReadNumber
MsgBox "Status Byte Register, Read: 0x" + Hex(varQueryResult)
```

The next program prints 0xD1 and clears bit 6 (RQS) and bit 4 (MAV) of the Status Byte Register. The difference in the output value between this example and the previous one is the value of bit 6 (weight = 64). Bit 6 is set when the first enabled summary bit is set and is cleared when the Status Byte Register is read by the serial poll command.

Example

The following example uses the resource session object’s ReadSTB method to read the contents of the oscilloscope’s Status Byte Register.

```
varQueryResult = myScope.IO.ReadSTB
MsgBox "Status Byte Register, Serial Poll: 0x" + Hex(varQueryResult)
```

NOTE **Use Serial Polling to Read Status Byte Register**. Serial polling is the preferred method to read the contents of the Status Byte Register because it resets bit 6 and allows the next enabled event that occurs to generate a new SRQ interrupt.

Status Reporting

Service Request Enable Register (SRE)

Setting the Service Request Enable Register bits enable corresponding bits in the Status Byte Register. These enabled bits can then set RQS and MSS (bit 6) in the Status Byte Register.

Bits are set in the Service Request Enable Register using the *SRE command and the bits that are set are read with the *SRE? query.

Example

The following example sets bit 4 (MAV) and bit 5 (ESB) in the Service Request Enable Register.

```
myScope.WriteString "*SRE " + CStr(CInt("&H30"))
```

This example uses the decimal parameter value of 48, the string returned by CStr(CInt("&H30")), to enable the oscilloscope to generate an SRQ interrupt under the following conditions:

- When one or more bytes in the Output Queue set bit 4 (MAV).
- When an enabled event in the Standard Event Status Register generates a summary bit that sets bit 5 (ESB).

Status Reporting

Trigger Event Register (TER)

This register sets the TRG bit in the status byte when a trigger event occurs.

The TER event register stays set until it is cleared by reading the register or using the *CLS command. If your application needs to detect multiple triggers, the TER event register must be cleared after each one.

If you are using the Service Request to interrupt a program or controller operation, you must clear the event register each time the trigger bit is set.

Status Reporting

Output Queue

The output queue stores the oscilloscope-to-controller responses that are generated by certain instrument commands and queries. The output queue generates the Message Available summary bit when the output queue contains one or more bytes. This summary bit sets the MAV bit (bit 4) in the Status Byte Register.

When using the Agilent VISA COM library, the output queue may be read with the FormattedIO488 object's ReadString, ReadNumber, ReadList, or ReadIEEEBlock methods.

Status Reporting

Message Queue

The message queue contains the text of the last message written to the advisory line on the screen of the oscilloscope. The length of the oscilloscope's message queue is 1. Note that messages sent with the :SYSTem:DSP command do not set the MSG status bit in the Status Byte Register.

Status Reporting

(Standard) Event Status Register (ESR)

The (Standard) Event Status Register (ESR) monitors the following oscilloscope status events:

- PON - Power On
- URQ - User Request
- CME - Command Error
- EXE - Execution Error
- DDE - Device Dependent Error
- QYE - Query Error
- RQC - Request Control
- OPC - Operation Complete

When one of these events occur, the event sets the corresponding bit in the register. If the bits are enabled in the Standard Event Status Enable Register, the bits set in this register generate a summary bit to set bit 5 (ESB) in the Status Byte Register.

You can read the contents of the Standard Event Status Register and clear the register by sending the *ESR? query. The value returned is the total bit weights of all of the bits that are set at the present time.

Example

The following example uses the *ESR query to read the contents of the Standard Event Status Register.

```
myScope.WriteString "*ESR?"
varQueryResult = myScope.ReadNumber
MsgBox "Standard Event Status Register: 0x" + Hex(varQueryResult)
```

If bit 4 (weight = 16) and bit 5 (weight = 32) are set, the program prints the sum of the two weights.

Status Reporting

(Standard) Event Status Enable Register (ESE)

To allow any of the (Standard) Event Status Register (ESR) bits to generate a summary bit, you must first enable that bit. Enable the bit by using the *ESE (Event Status Enable) common command to set the corresponding bit in the (Standard) Event Status Enable Register (ESE).

Set bits are read with the *ESE? query.

Example

Suppose your application requires an interrupt whenever any type of error occurs. The error related bits in the (Standard) Event Status Register are bits 2 through 5 (hexadecimal value 0x3C). Therefore, you can enable any of these bits to generate the summary bit by sending:

```
myScope.WriteString "*ESE " + CStr(CInt("&H3C"))
```

Whenever an error occurs, it sets one of these bits in the (Standard) Event Status Register. Because all the error related bits are enabled, a summary bit is generated to set bit 5 (ESB) in the Status Byte Register.

If bit 5 (ESB) in the Status Byte Register is enabled (via the *SRE command), an SRQ service request interrupt is sent to the controller PC.

NOTE **Disabled (Standard) Event Status Register bits respond but do not generate a summary bit.** (Standard) Event Status Register bits that are not enabled still respond to their corresponding conditions (that is, they are set if the corresponding event occurs). However, because they are not enabled, they do not generate a summary bit to the Status Byte Register.

Status Reporting

Error Queue

As errors are detected, they are placed in an error queue. This queue is first in, first out. If the error queue overflows, the last error in the queue is replaced with error 350, Queue overflow. Any time the queue overflows, the least recent errors remain in the queue, and the most recent error is discarded. The length of the oscilloscope’s error queue is 30 (29 positions for the error messages, and 1 position for the Queue overflow message).

The error queue is read with the :SYSTem:ERRor? query. Executing this query reads and removes the oldest error from the head of the queue, which opens a position at the tail of the queue for a new error. When all the errors have been read from the queue, subsequent error queries return "0, No error".

The error queue is cleared when:

- the instrument is powered up,
- the instrument receives the *CLS common command, or

- the last item is read from the error queue.

Status Reporting

Operation Status Event Register (:OPERegister[:EVENt])

This register hosts the RUN bit (bit 3), the WAIT TRIG bit (bit 5), and the OVLR bit (bit 11).

- The RUN bit is set whenever the instrument goes from a stop state to a single or running state.
- The WAIT TRIG bit is set by the Trigger Armed Event Register and indicates that the trigger is armed.
- The OVLR bit is set whenever a 50Ω input overload occurs.
- If any of these bits are set, the OPER bit (bit 7) of the Status Byte Register is set. The Operation Status Event Register is read and cleared with the :OPERegister[:EVENt]? query. The register output is enabled or disabled using the mask value supplied with the OPEE command.

Status Reporting

Operation Status Condition Register (:OPERegister:CONDition)

This register hosts the RUN bit (bit 3), the WAIT TRIG bit (bit 5), the OVLR bit (bit 11), and the HWE bit (bit 12).

- The :OPERegister:CONDition? query returns the value of the Operation Status Condition Register.
- The HWE bit (bit 12) comes from the Hardware Event Registers.
- The RUN bit is set whenever the instrument is not stopped.
- The WAIT TRIG bit is set by the Trigger Armed Event Register and indicates that the trigger is armed.
- The OVLR bit is set whenever a 50Ω input overload occurs.

Status Reporting

Arm Event Register (AER)

This register sets bit 5 (Wait Trig bit) in the Operation Status Register and the OPER bit (bit 7) in the Status Byte Register when the instrument becomes armed.

The ARM event register stays set until it is cleared by reading the register with the AER? query or using the *CLS command. If your application needs to detect multiple triggers, the ARM event register must be cleared after each one.

If you are using the Service Request to interrupt a program or controller operation when the trigger bit is set, then you must clear the event register after each time it has been set.

Status Reporting

Hardware Event Event Register (:HWERegister[:EVENt])

This register hosts the Bat On bit (bit 0).

- The Bat On bit is set whenever the instrument is operating on battery power.

Status Reporting

Hardware Event Condition Register (:HWERegister:CONDition)

This register hosts the Bat On bit (bit 0) and the PLL LOCKED bit (bit 12).

- The :HWERegister:CONDition? query returns the value of the Hardware Event Condition Register.
- The PLL LOCKED bit (bit 12) is for internal use and is not intended for general use.
- The Bat On bit is set whenever the instrument is operating on battery power.

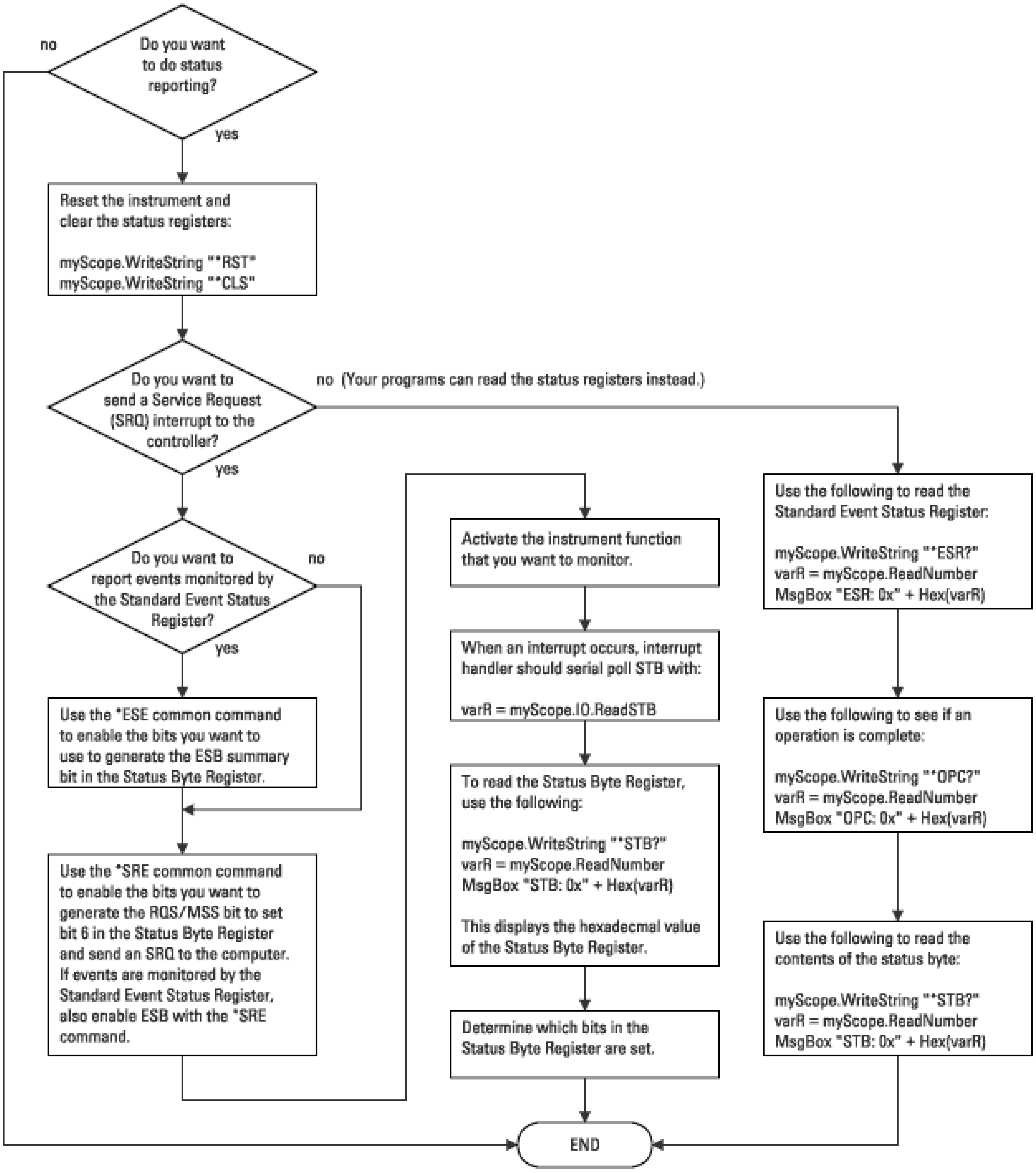
Status Reporting

Clearing Registers and Queues

The *CLS common command clears all event registers and all queues except the output queue. If *CLS is sent immediately after a program message terminator, the output queue is also cleared.

Status Reporting

Status Reporting Decision Chart



Do you want to do status reporting?

Reset the instrument and clear the status registers:

```
myScope.WriteString "*RST"
myScope.WriteString "*CLS"
```

Do you want to send a Service Request (SRQ) interrupt to the controller?

Do you want to report events monitored by the Standard Event Status Register?

Use the *ESE common command to enable the bits you want to use to generate the ESB summary bit in the Status Byte Register.

Use the *SRE common command to enable the bits you want to generate the RQS/MSS bit to set bit 6 in the Status Byte Register and send an SRQ to the computer. If events are monitored by the Standard Event Status Register, also enable ESB with the *SRE command.

Activate the instrument function that you want to monitor.

When an interrupt occurs, interrupt handler should serial poll STB with:

```
varR = myScope.IO.ReadSTB
```

To read the Status Byte Register, use the following:

```
myScope.WriteString "*STB?"
varR = myScope.ReadNumber
MsgBox "STB: 0x" + Hex(varR)
```

This displays the hexadecmal value of the Status Byte Register.

Determine which bits in the Status Byte Register are set.

Use the following to read the Standard Event Status Register:

```
myScope.WriteString "*ESR?"
varR = myScope.ReadNumber
MsgBox "ESR: 0x" + Hex(varR)
```

Use the following to see if an operation is complete:

```
myScope.WriteString "*OPC?"
varR = myScope.ReadNumber
MsgBox "OPC: 0x" + Hex(varR)
```

Use the following to read the contents of the status byte:

```
myScope.WriteString "*STB?"
varR = myScope.ReadNumber
MsgBox "STB: 0x" + Hex(varR)
```

More About Oscilloscope Commands

Command Classifications
Valid Command/Query Strings
Query Return Values
All Oscilloscope Commands Are Sequential

More About Oscilloscope Commands

Command Classifications

To help you use existing programs with the 6000 Series oscilloscopes, or use current programs with the next generation of oscilloscopes, commands are classified by the following categories:

- C Core Commands
- N Non-Core Commands
- O Obsolete Commands

Command Classifications

C Core Commands

Core commands are a common set of commands that provide basic oscilloscope functionality on this oscilloscope and future Agilent 6000 Series oscilloscopes. Core commands are unlikely to be modified in the future. If you restrict your programs to core commands, the programs should work across product offerings in the future, assuming appropriate programming methods are employed.

Command Classifications

N Non-Core Commands

Non-core commands are commands that provide specific features, but are not universal across all oscilloscope models. Non-core commands may be modified or deleted in the future. With a command structure as complex as the one for 6000 Series oscilloscopes, some evolution over time is inevitable. Agilent's intent is to continue to expand command subsystems, such as the rich and evolving trigger feature set.

Command Classifications

O Obsolete Commands

Obsolete commands are older forms of commands that are provided to reduce customer rework for existing systems and programs. Generally, these commands are mapped onto some of the Core and Non-core commands, but may not strictly have the same behavior as the new command. None of the obsolete commands are guaranteed to remain functional in future products. New systems and programs should use the Core (and Non-core) commands. Obsolete commands are listed in:

- *Obsolete and Discontinued Commands*
- As well as: *Commands A-Z*

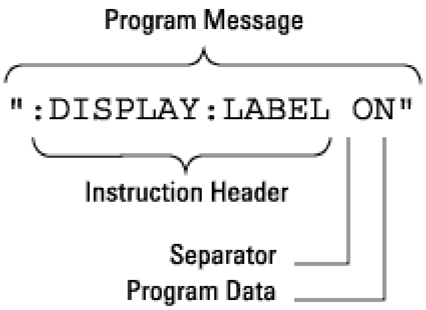
More About Oscilloscope Commands

Valid Command/Query Strings

Valid Command/Query Strings

Program Message Syntax

To program the instrument remotely, you must understand the command format and structure expected by the instrument. The IEEE 488.2 syntax rules govern how individual elements such as headers, separators, program data, and terminators may be grouped together to form complete instructions. Syntax definitions are also given to show how query responses are formatted. The following figure shows the main syntactical parts of a typical program statement.



Instructions (both commands and queries) normally appear as a string embedded in a statement of your host language, such as Visual Basic or C/C++. The only time a parameter is not meant to be expressed as a string is when the instruction's syntax definition specifies <block data>, such as <learn string>. There are only a few instructions that use block data.

Program messages can have long or short form commands (and data in some cases — see Long Form to Short Form Truncation Rules), and upper and/or lower case ASCII characters may be used. (Query responses, however, are always returned in upper case.)

Instructions are composed of two main parts:

- The header, which specifies the command or query to be sent.
- The program data, which provide additional information needed to clarify the meaning of the instruction.

Instruction Header

The instruction header is one or more mnemonics separated by colons (:) that represent the operation to be performed by the instrument. The Command Tree illustrates how all the mnemonics can be joined together to form a complete header.

":DISPlay:LABel ON" is a command. Queries are indicated by adding a question mark (?) to the end of the header, for example, ":DISPlay:LABel?". Many instructions can be used as either commands or queries, depending on whether or not you have included the question mark. The command and query forms of an instruction usually have different program data. Many queries do not use any program data.

There are three types of headers:

- Simple Command Headers
- Compound Command Headers
- Common Command Headers

White Space (Separator)

White space is used to separate the instruction header from the program data. If the instruction does not require any program data parameters, you do not need to include any white space. White space is defined as one or more space characters. ASCII defines a space to be character 32 (in decimal).

Program Data

Program data are used to clarify the meaning of the command or query. They provide necessary information, such as whether a function should be on or off, or which waveform is to be displayed. Each instruction’s syntax definition shows the program data, as well as the values they accept. Program Data Syntax Rules describes all of the general rules about acceptable values.

When there is more than one data parameter, they are separated by commas(,). Spaces can be added around the commas to improve readability.

Program Message Terminator

The program instructions within a data message are executed after the program message terminator is received. The terminator may be either an NL (New Line) character, an EOI (End-Or-Identify) asserted in the GPIB interface, or a combination of the two. Asserting the EOI sets the EOI control line low on the last byte of the data message. The NL character is an ASCII linefeed (decimal 10).

NOTE **New Line Terminator Functions**. The NL (New Line) terminator has the same function as an EOS (End Of String) and EOT (End Of Text) terminator.

Program Message Syntax

Long Form to Short Form Truncation Rules

To get the short form of a command/keyword:

- When the command/keyword is longer than four characters, use the first four characters of the command/keyword unless the fourth character is a vowel; when the fourth character is a vowel, use the first three characters of the command/keyword.
- When the command/keyword is four or fewer characters, use all of the characters.

Long Form	Short form
RANGe	RANG

PATTern	PATT
TIMebase	TIM
DELay	DEL
TYPE	TYPE

In the oscilloscope programmer's documentation, the short form of a command is indicated by uppercase characters.

Programs written in long form are easily read and are almost self-documenting. The short form syntax conserves the amount of controller memory needed for program storage and reduces I/O activity.

Program Message Syntax

Simple Command Headers

Simple command headers contain a single mnemonic. :AUToscale and :DIGitize are examples of simple command headers typically used in the oscilloscope. The syntax is:

```
<program mnemonic><terminator>
```

Simple command headers must occur at the beginning of a program message; if not, they must be preceded by a colon.

When program data must be included with the simple command header (for example, :DIGitize CHANnel1), white space is added to separate the data from the header. The syntax is:

```
<program mnemonic><separator><program data><terminator>
```

Program Message Syntax

Compound Command Headers

Compound command headers are a combination of two or more program mnemonics. The first mnemonic selects the subsystem, and the second mnemonic selects the function within that subsystem. The mnemonics within the compound message are separated by colons. For example, to execute a single function within a subsystem:

```
:<subsystem>:<function><separator><program data><terminator>
```

For example, :CHANnel1:BWLimit ON

Program Message Syntax

Common Command Headers

Common command headers control IEEE 488.2 functions within the instrument (such as clear status). Their syntax is:

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

No space or separator is allowed between the asterisk (*) and the command header. *CLS is an example of a common command header.

Program Message Syntax

Program Data Syntax Rules

Program data is used to convey a parameter information related to the command header. At least one space must separate the command header or query header from the program data.

```
<program mnemonic><separator><data><terminator>
```

When a program mnemonic or query has multiple program data, a comma separates sequential program data.

```
<program mnemonic><separator><data>,<data><terminator>
```

For example, :MEASure:DELay CHANnel1,CHANnel2 has two program data: CHANnel1 and CHANnel2.

Two main types of program data are used in commands: character and numeric.

Character Program Data

Character program data is used to convey parameter information as alpha or alphanumeric strings. For example, the :TIMebase:MODE command can be set to normal, delayed, XY, or ROLL. The character program data in this case may be MAIN, WINDow, XY, or ROLL. The command :TIMebase:MODE WINDow sets the time base mode to delayed.

The available mnemonics for character program data are always included with the commands’s syntax definition.

When sending commands, you may either the long form or short form (if one exists). Uppercase and lowercase letters may be mixed freely.

When receiving query responses, uppercase letters are used exclusively.

Numeric Program Data

Some command headers require program data to be expressed numerically. For example, :TIMebase:RANGe requires the desired full scale range to be expressed numerically.

For numeric program data, you have the option of using exponential notation or using suffix multipliers to indicate the numeric value. The following numbers are all equal:

28 = 0.28E2 = 280e-1 = 28000m = 0.028K = 28e-3K.

When a syntax definition specifies that a number is an integer, that means that the number should be whole. Any fractional part will be ignored, truncating the number. Numeric data parameters accept fractional values are called real numbers.

All numbers must be strings of ASCII characters. Thus, when sending the number 9, you would send a byte representing the ASCII code for the character 9 (which is 57). A three-digit number like 102 would take up three bytes (ASCII codes 49, 48, and 50). This is handled automatically when you include the entire instruction in a string.

Valid Command/Query Strings

Command Tree

The command tree shows all of the commands and the relationships of the commands to each other. The IEEE 488.2 common commands are not listed as part of the command tree because they do not affect the position of the parser within the tree. When a program message terminator (<NL>, linefeed-ASCII decimal 10) or a leading colon (:) is sent to the instrument, the parser is set to the root of the command tree.

: (root)

- :ACQuire
    - :AALias
    - :COMPlete
    - :COUNt
    - :DAALias
    - :MODE
    - :POINts
    - :RSIGnal
    - :SRATe
    - :TYPE
- :ACTivity
- :AER (Arm Event Register)
- :AUToscale
    - :AMODE
    - :CHANnels
- :BLANk
- :BUS<n>
    - :BIT<m>
    - :BITS
    - :CLEar
    - :DISPlay
    - :LABel
    - :MASK
- :CALibrate
    - :DATE
    - :LABel
    - :STARt
    - :STATus
    - :SWITch
    - :TEMPerature
    - :TIME

- :CDISplay
- :CHANnel<n>
    - :BWLimit
    - :COUPling
    - :DISPlay
    - :IMPedance
    - :INVert
    - :LABel
    - :OFFSet
    - :PROBe
        - :ID
        - :SKEW
        - :STYPe
    - :PROTection
    - :RANGe
    - :SCALe
    - :UNITs
    - :VERNier
- :DIGital<n>
    - :DISPlay
    - :LABel
    - :POSition
    - :SIZE
    - :THReshold
- :DIGitize
- :DISPlay
    - :CLEar
    - :DATA
    - :LABel
    - :LABList
    - :PERSistence
    - :SOURce
    - :VECTors
- :EXTernal
    - :BWLimit
    - :IMPedance
    - :PROBe
        - :ID

        - :STYPe
    - :PROTection
    - :RANGe
    - :UNITs
- :FUNCtion
    - :CENTer
    - :DISPlay
    - :OFFSet
    - :OPERation
    - :RANGe
    - :REFerence
    - :SCALe
    - :SOURce
    - :SPAN
    - :WINDow
- :HARDcopy
    - :FACTors
    - :FFEed
    - :FILename
    - :FORMat
    - :IGColors
    - :PALette
    - :PDRiver
- :HWEenable (Hardware Event Enable Register)
- :HWERegister
    - :CONDition (Hardware Event Condition Register)
    - [:EVENt] (Hardware Event Event Register)
- :MARKer
    - :MODE
    - :X1Position
    - :X1Y1source
    - :X2Position
    - :X2Y2source
    - :XDELta
    - :Y1Position
    - :Y2Position
    - :YDELta
- :MEASure

        - :DISPlay
        - :SIZE
        - :THReshold
- :RUN
- :SBUS
    - :CAN
        - :COUNt
            - :ERRor
            - :OVERload
            - :RESet
            - :TOTal
            - :UTILization
    - :DISPlay
    - :IIC
        - :WIDTh
    - :LIN
        - :PARity
    - :MODE
    - :SPI
        - :ASIZe
- :SERial
- :SINGle
- :STATus
- :STOP
- :SYSTem
    - :DATE
    - :DSP
    - :ERRor
    - :LOCK
    - :SETup
    - :TIME
- :TER (Trigger Event Register)
- :TIMebase
    - :MODE
    - :POSition
    - :RANGe
    - :REFClock
    - :REFerence

    - :SCALe
    - :VERNier
    - :WINDow
        - :POSition
        - :RANGe
        - :SCALe
- :TRIGger
    - :HFReject
    - :HOLDoff
    - :MODE
    - :NREJect
    - :PATTern
    - :SWEep
    - :CAN
        - :ACKNowledge
        - :PATTern
            - :DATA
                - :LENGth
            - :ID
                - :MODE
        - :SAMPlepoint
        - :SIGNal
            - :BAUDrate
            - :DEFinition
        - :SOURce
        - :TRIGger
    - :DURation
        - :COUNt
        - :IDLE
        - :SLOPe
    - :EBURst
        - :GREaterthan
        - :LESSthan
        - :PATTern
        - :QUALifier
        - :RANGe
    - [:EDGE]
        - :COUPling

  - :LEVel
  - :REJect
  - :SLOPe
  - :SOURce
- :GLITch
  - :GREaterthan
  - :LESSthan
  - :LEVel
  - :POLarity
  - :QUALifier
  - :RANGe
  - :SOURce
- :HFReject
- :HOLDoff
- :IIC
  - :PATTern
    - :ADDRess
    - :DATA
    - :DATa2
  - :SOURce
    - :CLOCk
    - :DATA
  - :TRIGger
    - :QUALifier
    - [:TYPE]
- :LIN
  - :ID
  - :SAMPlepoint
  - :SIGNal
    - :BAUDrate
    - :DEFinition
  - :SOURce
  - :STANdard
  - :SYNCbreak
  - :TRIGger
- :MODE
- :NREJect
- :PATTern

  - :SEQuence
    - :COUNt
    - :EDGE
    - :FIND
    - :PATTern
    - :RESet
    - :TIMer
    - :TRIGger
  - :SPI
    - :CLOCk
      - :SLOPe
      - :TIMeout
    - :FRAMing
    - :PATTern
      - :DATA
      - :WIDTh
    - :SOURce
      - :CLOCk
      - :DATA
      - :FRAMe
  - :SWEep
  - :TV
    - :LINE
    - :MODE
    - :POLarity
    - :SOURce
    - :STANdard
    - :TVMode
  - :USB
    - :SOURce
      - :DMINus
      - :DPLus
    - :SPEed
    - :TRIGger
- :VIEW
- :WAVeform
  - :BYTeorder
  - :COUNt

- :DATA
- :FORMat
- :POINts
  - :MODE
- :PREamble
- :SOURce
- :TYPE
- :UNSigned
- :VIEW
- :XINCrement
- :XORigin
- :XREFerence
- :YINCrement
- :YORigin
- :YREFerence

Common Commands (IEEE 488.2)

- *CLS
- *ESE
- *ESR
- *IDN
- *LRN
- *OPC
- *OPT
- *RCL
- *RST
- *SAV
- *SRE
- *STB
- *TRG
- *TST
- *WAI

Valid Command/Query Strings

Duplicate Mnemonics

Identical function mnemonics can be used in more than one subsystem. For example, the function mnemonic RANGe may be used to change the vertical range or to change the horizontal range:

```
:CHANnel1:RANGe .4
```

Sets the vertical range of channel 1 to 0.4 volts full scale.

```
:TIMebase:RANGe 1
```

Sets the horizontal time base to 1 second full scale.

:CHANnel1 and :TIMebase are subsystem selectors and determine which range is being modified.

Valid Command/Query Strings

Tree Traversal Rules and Multiple Commands

Command headers are created by traversing down the Command Tree. A legal command header would be :TIMebase:RANGe. This is referred to as a *compound header*. A compound header is a header made of two or more mnemonics separated by colons. The mnemonic created contains no spaces.

The following rules apply to traversing the tree:

- A leading colon (<NL> or EOI true on the last byte) places the parser at the root of the command tree. A leading colon is a colon that is the first character of a program header. Executing a subsystem command lets you access that subsystem until a leading colon or a program message terminator (<NL>) or EOI true is found.
- In the command tree, use the last mnemonic in the compound header as the reference point (for example, RANGe). Then find the last colon above that mnemonic (TIMebase:). That is the point where the parser resides. Any command below that point can be sent within the current program message without sending the mnemonics which appear above them (for example, POSition).

The output statements in the examples are written using the Agilent VISA COM library in Visual Basic. The quoted string is placed on the bus, followed by a carriage return and linefeed (CRLF).

To execute more than one function within the same subsystem, separate the functions with a semicolon (;):

```
:<subsystem>:<function><separator><data>;<function><separator><data><terminator>
```

For example:

```
myScope.WriteString ":TIMebase:RANGe 0.5;POSition 0"
```

NOTE The colon between TIMebase and RANGe is necessary because TIMebase:RANGe is a compound command. The semicolon between the RANGe command and the POSition command is the required program message unit separator. The POSition command does not need TIMebase preceding it because the TIMebase:RANGe command sets the parser to the TIMebase node in the tree.

Example 2: Program Message Terminator Sets Parser Back to Root

```
myScope.WriteString ":TIMebase:REFerence CENTer;POSition 0.00001"
```

or

```
myScope.WriteString ":TIMebase:REFerence CENTer"
myScope.WriteString ":TIMebase:POSition 0.00001"
```

NOTE In the first line of example 2, the subsystem selector is implied for the POSition command in the compound command. The POSition command must be in the same program message as the REFerence command because the program message terminator places the parser back at the root of the command tree.

A second way to send these commands is by placing TIMebase: before the POSition command as shown in the second part of example 2. The space after POSition is required.

Example 3: Selecting Multiple Subsystems

You can send multiple program commands and program queries for different subsystems on the same line by separating each command with a semicolon. The colon following the semicolon enables you to enter a new subsystem. For example:

```
<program mnemonic><data>;:<program mnemonic><data><terminator>
```

For example:

```
myScope.WriteString ":TIMebase:REFerence CENTer;:DISPlay:VECTors ON"
```

NOTE The leading colon before DISPlay:VECTors ON tells the parser to go back to the root of the command tree. The parser can then see the DISPlay:VECTors ON command. The space between REFerence and CENter is required; so is the space between VECTors and ON.

Multiple commands may be any combination of compound and simple commands.

More About Oscilloscope Commands

Query Return Values

Command headers immediately followed by a question mark (?) are queries. Queries are used to get results of measurements made by the instrument or to find out how the instrument is currently configured.

After receiving a query, the instrument interrogates the requested function and places the answer in its output queue. The answer remains in the output queue until it is read or another command is issued.

When read, the answer is transmitted across the bus to the designated listener (typically a controller). For example, the query :TIMebase:RANGe? places the current time base setting in the output queue. When using the Agilent VISA COM library in Visual Basic, the controller statements:

```
Dim strQueryResult As String
myScope.WriteString ":TIMebase:RANGe?"
strQueryResult = myScope.ReadString
```

pass the value across the bus to the controller and place it in the variable strQueryResult.

NOTE **Read Query Results Before Sending Another Command**. Sending another command or query before reading the result of a query clears the output buffer (the current response) and places a Query INTERRUPTED error in the error queue.

Infinity Representation

The representation of infinity is +9.9E+37. This is also the value returned when a measurement cannot be made.

More About Oscilloscope Commands

All Oscilloscope Commands Are Sequential

IEEE 488.2 makes the distinction between sequential and overlapped commands:

- *Sequential commands* finish their task before the execution of the next command starts.
- *Overlapped commands* run concurrently. Commands following an overlapped command may be started before the overlapped command is completed.

All of the oscilloscope commands are sequential.

Agilent 6000 Series Oscilloscopes Programmer's Reference

Programming Examples

SICL Example in C
VISA Example in C
VISA Example in Visual Basic
VISA COM Example in Visual Basic

Example programs are ASCII text files that can be cut from the help file and pasted into your favorite text editor.

Programming Examples

SICL Example in C

```
/*
 * Agilent SICL Example in C
 * ------------------------------------------------------------------
 * This program illustrates most of the commonly-used programming
 * features of the Agilent DSO/MSO6000-series oscilloscopes.
 * This program is to be built as a WIN32 console application.
 * Edit the DEVICE_ADDRESS line to specify the address of the
 * applicable device.
 */

#include <stdio.h>            /* For printf(). */
#include "sicl.h"             /* SICL routines. */

/* #define DEVICE_ADDRESS "gpib0,7" */                      /* GPIB */
/* #define DEVICE_ADDRESS "lan[a-mso6102-90541]:inst0" */   /* LAN */
#define DEVICE_ADDRESS "usb0[2391::5970::30D3090541::0]"    /* USB */

#define WAVE_DATA_SIZE 5000
#define TIMEOUT        5000
#define SETUP_STR_SIZE 3000
#define IMG_SIZE     300000

/* Function prototypes */
void initialize(void);          /* Initialize the oscilloscope. */
void extra(void);               /* Miscellaneous commands not executed,
                                   shown for reference purposes. */
void capture(void);             /* Digitize data from oscilloscope. */
void analyze(void);             /* Make some measurements. */
void get_waveform(void);        /* Download waveform data from oscilloscope. */
void save_waveform(void);       /* Save waveform data to a file. */
void retrieve_waveform(void);   /* Load waveform data from a file. */

/* Global variables */
INST id;                                       /* Device session ID. */
char buf[256] = { 0 };                         /* Buffer for IDN string. */
unsigned char waveform_data[WAVE_DATA_SIZE];   /* Array for waveform data. */
double preamble[10];                           /* Array for preamble. */

void main(void)
{
   /* Install a default SICL error handler that logs an error message
    * and exits.  On Windows 98SE or Windows Me, view messages with
    * the SICL Message Viewer.  For Windows 2000 or XP, use the Event
    * Viewer.
    */
   ionerror(I_ERROR_EXIT);

   /* Open a device session using the DEVICE_ADDRESS */
   id = iopen(DEVICE_ADDRESS);

   if (id == 0)
   {
      printf ("Oscilloscope iopen failed!\n");
   }
   else
   {
      printf ("Oscilloscope session initialized!\n");

      /* Set the I/O timeout value for this session to 5 seconds. */
      itimeout(id, TIMEOUT);

      /* Clear the interface. */
      iclear(id);
      iremote(id);
   }

   initialize();

   /* The extras function contains miscellaneous commands that do not
    * need to be executed for the proper operation of this example.
    * The commands in the extras function are shown for reference
    * purposes only.
    */
   /* extra(); */   /* <-- Uncomment to execute the extra function */

   capture();

   analyze();

   /* Close the device session to the instrument. */
   iclose(id);
   printf ("Program execution is complete...\n");

   /* For WIN16 programs, call _siclcleanup before exiting to release
    * resources allocated by SICL for this application.  This call is
    * a no-op for WIN32 programs.
    */
   _siclcleanup();
}

/*
 * initialize
 * ------------------------------------------------------------------
 * This function initializes both the interface and the oscilloscope
 * to a known state.
 */

void initialize (void)
{
   /* RESET - This command puts the oscilloscope in a known state.
    * Without this command, the oscilloscope settings are unknown.
    * This command is very important for program control.
    *
    * Many of the following initialization commands are initialized
    * by this command.  It is not necessary to reinitialize them
    * unless you want to change the default setting.
    */
   iprintf(id, "*RST\n");

   /* Write the *IDN? string and send an EOI indicator, then read
    * the response into buf.
   ipromptf(id, "*IDN?\n", "%t", buf);
   printf("%s\n", buf);
    */

   /* AUTOSCALE - This command evaluates all the input signals and
    * sets the correct conditions to display all of the active signals.
    */
   iprintf(id, ":AUTOSCALE\n");

   /* CHANNEL_PROBE - Sets the probe attenuation factor for the
    * selected channel.  The probe attenuation factor may be from
    * 0.1 to 1000.
    */
   iprintf(id, ":CHAN1:PROBE 10\n");

   /* CHANNEL_RANGE - Sets the full scale vertical range in volts.
    * The range value is eight times the volts per division.
    */
   iprintf(id, ":CHANNEL1:RANGE 8\n");

   /* TIME_RANGE - Sets the full scale horizontal time in seconds.
    * The range value is ten times the time per division.
    */
   iprintf(id, ":TIM:RANG 2e-3\n");

   /* TIME_REFERENCE - Possible values are LEFT and CENTER:
    *  - LEFT sets the display reference one time division from the left.
    *  - CENTER sets the display reference to the center of the screen.
    */
   iprintf(id, ":TIMEBASE:REFERENCE CENTER\n");

   /* TRIGGER_SOURCE - Selects the channel that actually produces the
    * TV trigger.  Any channel can be selected.
    */
   iprintf(id, ":TRIGGER:TV:SOURCE CHANNEL1\n");

   /* TRIGGER_MODE - Set the trigger mode to, EDGE, GLITch, PATTern,
    * CAN, DURation, IIC, LIN, SEQuence, SPI, TV, or USB.
    */
   iprintf(id, ":TRIGGER:MODE EDGE\n");

   /* TRIGGER_EDGE_SLOPE - Set the slope of the edge for the trigger
    * to either POSITIVE or NEGATIVE.
    */
   iprintf(id, ":TRIGGER:EDGE:SLOPE POSITIVE\n");
}

/*
```

```
 * extra
 * ------------------------------------------------------------------
 * The commands in this function are not executed and are shown for
 * reference purposes only.  To execute these commands, call this
 * function from main.
 */

void extra (void)
{
   /* RUN_STOP (not executed in this example):
    *  - RUN starts the acquisition of data for the active waveform
    *    display.
    *  - STOP stops the data acquisition and turns off AUTOSTORE.
    */
   iprintf(id, ":RUN\n");
   iprintf(id, ":STOP\n");

   /* VIEW_BLANK (not executed in this example):
    *  - VIEW turns on (starts displaying) an active channel or pixel
    *    memory.
    *  - BLANK turns off (stops displaying) a specified channel or
    *    pixel memory.
    */
   iprintf(id, ":BLANK CHANNEL1\n");
   iprintf(id, ":VIEW CHANNEL1\n");

   /* TIME_MODE (not executed in this example) - Set the time base
    * mode to MAIN, DELAYED, XY or ROLL.
    */
   iprintf(id, ":TIMEBASE:MODE MAIN\n");
}

/*
 * capture
 * ------------------------------------------------------------------
 * This function prepares the scope for data acquisition and then
 * uses the DIGITIZE MACRO to capture some data.
 */

void capture (void)
{
   /* AQUIRE_TYPE - Sets the acquisition mode.  There are three
    * acquisition types NORMAL, PEAK, or AVERAGE.
    */
   iprintf(id, ":ACQUIRE:TYPE NORMAL\n");

   /* AQUIRE_COMPLETE - Specifies the minimum completion criteria
    * for an acquisition.  The parameter determines the percentage
    * of time buckets needed to be "full" before an acquisition is
    * considered to be complete.
    */
   iprintf(id, ":ACQUIRE:COMPLETE 100\n");

   /* DIGITIZE - Used to acquire the waveform data for transfer over
    * the interface.  Sending this command causes an acquisition to
    * take place with the resulting data being placed in the buffer.
    */

   /* NOTE!  The use of the DIGITIZE command is highly recommended
    * as it will ensure that sufficient data is available for
    * measurement.  Keep in mind when the oscilloscope is running,
    * communication with the computer interrupts data acquisition.
    * Setting up the oscilloscope over the bus causes the data
    * buffers to be cleared and internal hardware to be reconfigured.
    * If a measurement is immediately requested there may not have
    * been enough time for the data acquisition process to collect
    * data and the results may not be accurate.  An error value of
    * 9.9E+37 may be returned over the bus in this situation.
    */
   iprintf(id, ":DIGITIZE CHAN1\n");
}

/*
 * analyze
 * ------------------------------------------------------------------
 * In this example we will do the following:
 *  - Save the system setup to a file for restoration at a later time.
 *  - Save the oscilloscope display to a file which can be printed.
 *  - Make single channel measurements.
 */

void analyze (void)
{
   double frequency, vpp;           /* Measurements. */
   double vdiv, off, sdiv, delay;   /* Values calculated from preamble data. */
   int i;                           /* Loop counter. */
   unsigned char setup_string[SETUP_STR_SIZE];   /* Array for setup string. */
   int setup_size;
   FILE *fp;
   unsigned char image_data[IMG_SIZE];         /* Array for image data. */
   int img_size;

   /* SAVE_SYSTEM_SETUP - The :SYSTEM:SETUP? query returns a program message
    * that contains the current state of the instrument.  Its format is a
    * definite-length binary block, for example, #800002204<setup string><NL>
    * where the setup string is 2204 bytes in length.
    */
   setup_size = SETUP_STR_SIZE;
   /* Query and read setup string. */
   ipromptf(id, ":SYSTEM:SETUP?\n", "%#b\n", &setup_size, setup_string);
   printf("Read setup string query (%d bytes).\n", setup_size);
   /* Write setup string to file. */
   fp = fopen ("c:\\scope\\config\\setup.dat", "wb");
   setup_size = fwrite(setup_string, sizeof(unsigned char), setup_size, fp);
   fclose (fp);
   printf("Wrote setup string (%d bytes) to file.\n", setup_size);

   /* RESTORE_SYSTEM_SETUP - Uploads a previously saved setup string to the
    * oscilloscope.
    */
   /* Read setup string from file. */
   fp = fopen ("c:\\scope\\config\\setup.dat", "rb");
   setup_size = fread (setup_string, sizeof(unsigned char), SETUP_STR_SIZE, fp);
   fclose (fp);
   printf("Read setup string (%d bytes) from file.\n", setup_size);
   /* Restore setup string. */
   iprintf(id, ":SYSTEM:SETUP #8%08d", setup_size);
   ifwrite(id, setup_string, setup_size, 1, &setup_size);
   printf("Restored setup string (%d bytes).\n", setup_size);

   /* IMAGE_TRANSFER - In this example we will query for the image
    * data with ":DISPLAY:DATA?" to read the data and save the data
    * to the file "image.dat" which you can then send to a printer.
    */
   itimeout(id, 30000);
   printf("Transferring image to c:\\scope\\data\\screen.bmp\n");
   img_size = IMG_SIZE;
   ipromptf(id, ":DISPLAY:DATA? BMP8bit, SCREEN, COLOR\n", "%#b\n", &img_size, image_data);
   printf("Read display data query (%d bytes).\n", img_size);
   /* Write image data to file. */
   fp = fopen ("c:\\scope\\data\\screen.bmp", "wb");
   img_size = fwrite(image_data, sizeof(unsigned char), img_size, fp);
   fclose (fp);
   printf("Wrote image data (%d bytes) to file.\n", img_size);
   itimeout(id, 5000);

   /* MEASURE - The commands in the MEASURE subsystem are used to
    * make measurements on displayed waveforms.
    */
   iprintf(id, ":MEASURE:SOURCE CHANNEL1\n");   /* Set source to measure. */

   /* Query for frequency. */
   ipromptf(id, ":MEASURE:FREQUENCY?\n", "%lf", &frequency);
   printf("The frequency is: %.4f kHz\n", frequency / 1000);

   /* Query for peak to peak voltage. */
   ipromptf(id, ":MEASURE:VPP?\n", "%lf", &vpp);
   printf("The peak to peak voltage is: %.2f V\n", vpp);

   /* WAVEFORM_DATA - Get waveform data from oscilloscope.
    */
   get_waveform();

   /* Make some calculations from the preamble data. */
   vdiv  = 32 * preamble [7];
   off   = preamble [8];
   sdiv  = preamble [2] * preamble [4] / 10;
   delay = (preamble [2] / 2) * preamble [4] + preamble [5];

   /* Print them out... */
   printf ("Scope Settings for Channel 1:\n");
   printf ("Volts per Division = %f\n", vdiv);
   printf ("Offset = %f\n", off);
   printf ("Seconds per Division = %f\n", sdiv);
   printf ("Delay = %f\n", delay);

   /* print out the waveform voltage at selected points */
   for (i = 0; i < 1000; i = i + 50)
      printf ("Data Point %4d = %6.2f Volts at %10f Seconds\n", i,
      ((float)waveform_data[i] - preamble[9]) * preamble[7] + preamble[8],
      ((float)i - preamble[6]) * preamble[4] + preamble[5]);

   save_waveform();       /* Save waveform data to disk. */
   retrieve_waveform();   /* Load waveform data from disk. */
}

/*
 * get_waveform
 * --------------------------------------------------------------------
 * This function transfers the data displayed on the oscilloscope to
 * the computer for storage, plotting, or further analysis.
 */

void get_waveform (void)
{
   int waveform_size;

   /* WAVEFORM_DATA - To obtain waveform data, you must specify the
    * WAVEFORM parameters for the waveform data prior to sending the
    * ":WAVEFORM:DATA?" query.
    *
    * Once these parameters have been sent, the ":WAVEFORM:PREAMBLE?"
    * query provides information concerning the vertical and horizontal
    * scaling of the waveform data.
    *
    * With the preamble information you can then use the ":WAVEFORM:DATA?"
    * query and read the data block in the correct format.
    */

   /* WAVE_FORMAT - Sets the data transmission mode for waveform data
    * output.  This command controls how the data is formatted when
```

```
     * sent from the oscilloscope and can be set to WORD or BYTE format.
     */
    iprintf(id, ":WAVEFORM:FORMAT BYTE\n");   /* Set waveform format to BYTE. */

    /* WAVE_POINTS - Sets the number of points to be transferred.  The
     * number of time points available is returned by the "ACQUIRE:POINTS?"
     * query.  This can be set to any binary fraction of the total time
     * points available.
     */
    iprintf(id, ":WAVEFORM:POINTS 1000\n");

    /* GET_PREAMBLE - The preamble contains all of the current WAVEFORM
     * settings returned in the form <preamble block><NL> where the
     * <preamble block> is:
     *    FORMAT     : int16 - 0 = BYTE, 1 = WORD, 2 = ASCII.
     *    TYPE       : int16 - 0 = NORMAL, 1 = PEAK DETECT, 2 = AVERAGE.
     *    POINTS     : int32 - number of data points transferred.
     *    COUNT      : int32 - 1 and is always 1.
     *    XINCREMENT : float64 - time difference between data points.
     *    XORIGIN    : float64 - always the first data point in memory.
     *    XREFERENCE : int32 - specifies the data point associated w/x-origin.
     *    YINCREMENT : float32 - voltage difference between data points.
     *    YORIGIN    : float32 - value of the voltage at center screen.
     *    YREFERENCE : int32 - data point where y-origin occurs.
     */
    printf("Reading preamble\n");
    ipromptf(id, ":WAVEFORM:PREAMBLE?\n", "%,10lf\n", preamble);
    /*
    printf("Preamble FORMAT: %e\n", preamble[0]);
    printf("Preamble TYPE: %e\n", preamble[1]);
    printf("Preamble POINTS: %e\n", preamble[2]);
    printf("Preamble COUNT: %e\n", preamble[3]);
    printf("Preamble XINCREMENT: %e\n", preamble[4]);
    printf("Preamble XORIGIN: %e\n", preamble[5]);
    printf("Preamble XREFERENCE: %e\n", preamble[6]);
    printf("Preamble YINCREMENT: %e\n", preamble[7]);
    printf("Preamble YORIGIN: %e\n", preamble[8]);
    printf("Preamble YREFERENCE: %e\n", preamble[9]);
    */

    /* QUERY_WAVE_DATA - Outputs waveform records to the controller
     * over the interface that is stored in a buffer previously
     * specified with the ":WAVEFORM:SOURCE" command.
     */
    iprintf(id, ":WAVEFORM:DATA?\n");   /* Query waveform data. */

    /* READ_WAVE_DATA - The wave data consists of two parts: the header,
     * and the actual waveform data followed by an New Line (NL) character.
     * The query data has the following format:
     *
     *    <header><waveform data block><NL>
     *
     * Where:
     *
     *    <header> = #800002048    (this is an example header)
     *
     * The "#8" may be stripped off of the header and the remaining
     * numbers are the size, in bytes, of the waveform data block.
     * The size can vary depending on the number of points acquired
     * for the waveform which can be set using the ":WAVEFORM:POINTS"
     * command.  You may then read that number of bytes from the
     * oscilloscope; then, read the following NL character to
     * terminate the query.
     */
    waveform_size = WAVE_DATA_SIZE;
    /* Read waveform data. */
    iscanf(id, "%#b\n", &waveform_size, waveform_data);
```

```
   if ( waveform_size == WAVE_DATA_SIZE )
   {
      printf("Waveform data buffer full: May not have received all points.\n");
   }
   else
   {
      printf("Reading waveform data... size = %d\n", waveform_size);
   }
}

/*
 * save_waveform
 * ------------------------------------------------------------------
 * This function saves the waveform data from the get_waveform
 * function to disk.  The data is saved to a file called "wave.dat".
 */

void save_waveform(void)
{
   FILE *fp;

   fp = fopen("c:\\scope\\data\\wave.dat", "wb");
   fwrite(preamble, sizeof(preamble[0]), 10, fp);   /* Write preamble. */
   /* Write actually waveform data. */
   fwrite(waveform_data, sizeof(waveform_data[0]), (int)preamble[2], fp);
   fclose (fp);
}

/*
 * retrieve_waveform
 * ------------------------------------------------------------------
 * This function retrieves previously saved waveform data from a
 * file called "wave.dat".
 */

void retrieve_waveform(void)
{
   FILE *fp;

   fp = fopen("c:\\scope\\data\\wave.dat", "rb");
   fread (preamble, sizeof(preamble[0]), 10, fp);   /* Read preamble. */
   /* Read the waveform data. */
   fread (waveform_data, sizeof(waveform_data[0]), (int)preamble[2], fp);
   fclose (fp);
}
```

VISA Example in C

```
/*
 * Agilent VISA Example in C
 * ------------------------------------------------------------------
 * This program illustrates most of the commonly-used programming
 * features of the Agilent DSO/MSO6000-series oscilloscopes.
 * This program is to be built as a WIN32 console application.
 * Edit the RESOURCE line to specify the address of the
 * applicable device.
 */

#include <stdio.h>            /* For printf(). */
#include <visa.h>             /* Agilent VISA routines. */
```

```
/* #define RESOURCE "GPIB0::7::INSTR" */                         /* GPIB */
/* #define RESOURCE "TCPIP0::a-mso6102-90541::inst0::INSTR" */   /* LAN */
#define RESOURCE "USB0::2391::5970::30D3090541::0::INSTR"        /* USB */

#define WAVE_DATA_SIZE 5000
#define TIMEOUT        5000
#define SETUP_STR_SIZE   3000
#define IMG_SIZE     300000

/* Function prototypes */
void initialize(void);          /* Initialize the oscilloscope. */
void extra(void);               /* Miscellaneous commands not executed,
                                   shown for reference purposes. */
void capture(void);             /* Digitize data from oscilloscope. */
void analyze(void);             /* Make some measurements. */
void get_waveform(void);        /* Download waveform data from oscilloscope. */
void save_waveform(void);       /* Save waveform data to a file. */
void retrieve_waveform(void);   /* Load waveform data from a file. */

/* Global variables */
ViSession defaultRM, vi;                       /* Device session ID. */
char buf[256] = { 0 };                         /* Buffer for IDN string. */
unsigned char waveform_data[WAVE_DATA_SIZE];   /* Array for waveform data. */
double preamble[10];                           /* Array for preamble. */

void main(void)
{
   /* Open session. */
   viOpenDefaultRM(&defaultRM);
   viOpen(defaultRM, RESOURCE, VI_NULL,VI_NULL, &vi);
   printf ("Oscilloscope session initialized!\n");

   /* Clear the interface. */
   viClear(vi);

   initialize();

   /* The extras function contains miscellaneous commands that do not
    * need to be executed for the proper operation of this example.
    * The commands in the extras function are shown for reference
    * purposes only.
    */
   /* extra(); */   /* <-- Uncomment to execute the extra function */

   capture();

   analyze();

   /* Close session */
   viClose(vi);
   viClose(defaultRM);
   printf ("Program execution is complete...\n");
}

/*
 * initialize
 * ------------------------------------------------------------------
 * This function initializes both the interface and the oscilloscope
 * to a known state.
 */

void initialize (void)
{
   /* RESET - This command puts the oscilloscope in a known state.
    * Without this command, the oscilloscope settings are unknown.
    * This command is very important for program control.
    *
    * Many of the following initialization commands are initialized
    * by this command.  It is not necessary to reinitialize them
    * unless you want to change the default setting.
    */
   viPrintf(vi, "*RST\n");

   /* Write the *IDN? string and send an EOI indicator, then read
    * the response into buf.
   viQueryf(vi, "*IDN?\n", "%t", buf);
   printf("%s\n", buf);
    */

   /* AUTOSCALE - This command evaluates all the input signals and
    * sets the correct conditions to display all of the active signals.
    */
   viPrintf(vi, ":AUTOSCALE\n");

   /* CHANNEL_PROBE - Sets the probe attenuation factor for the
    * selected channel.  The probe attenuation factor may be from
    * 0.1 to 1000.
    */
   viPrintf(vi, ":CHAN1:PROBE 10\n");

   /* CHANNEL_RANGE - Sets the full scale vertical range in volts.
    * The range value is eight times the volts per division.
    */
   viPrintf(vi, ":CHANNEL1:RANGE 8\n");

   /* TIME_RANGE - Sets the full scale horizontal time in seconds.
    * The range value is ten times the time per division.
    */
   viPrintf(vi, ":TIM:RANG 2e-3\n");

   /* TIME_REFERENCE - Possible values are LEFT and CENTER:
    *  - LEFT sets the display reference one time division from the left.
    *  - CENTER sets the display reference to the center of the screen.
    */
   viPrintf(vi, ":TIMEBASE:REFERENCE CENTER\n");

   /* TRIGGER_SOURCE - Selects the channel that actually produces the
    * TV trigger.  Any channel can be selected.
    */
   viPrintf(vi, ":TRIGGER:TV:SOURCE CHANNEL1\n");

   /* TRIGGER_MODE - Set the trigger mode to, EDGE, GLITch, PATTern,
    * CAN, DURation, IIC, LIN, SEQuence, SPI, TV, or USB.
    */
   viPrintf(vi, ":TRIGGER:MODE EDGE\n");

   /* TRIGGER_EDGE_SLOPE - Set the slope of the edge for the trigger
    * to either POSITIVE or NEGATIVE.
    */
   viPrintf(vi, ":TRIGGER:EDGE:SLOPE POSITIVE\n");
}

/*
 * extra
 * ------------------------------------------------------------------
 * The commands in this function are not executed and are shown for
 * reference purposes only.  To execute these commands, call this
 * function from main.
 */

void extra (void)
{
   /* RUN_STOP (not executed in this example):
    *  - RUN starts the acquisition of data for the active waveform
    *    display.
    *  - STOP stops the data acquisition and turns off AUTOSTORE.
    */
   viPrintf(vi, ":RUN\n");
   viPrintf(vi, ":STOP\n");

   /* VIEW_BLANK (not executed in this example):
    *  - VIEW turns on (starts displaying) an active channel or pixel
    *    memory.
    *  - BLANK turns off (stops displaying) a specified channel or
    *    pixel memory.
    */
   viPrintf(vi, ":BLANK CHANNEL1\n");
   viPrintf(vi, ":VIEW CHANNEL1\n");

   /* TIME_MODE (not executed in this example) - Set the time base
    * mode to MAIN, DELAYED, XY or ROLL.
    */
   viPrintf(vi, ":TIMEBASE:MODE MAIN\n");
}

/*
 * capture
 * --------------------------------------------------------------------
 * This function prepares the scope for data acquisition and then
 * uses the DIGITIZE MACRO to capture some data.
 */

void capture (void)
{
   /* AQUIRE_TYPE - Sets the acquisition mode.  There are three
    * acquisition types NORMAL, PEAK, or AVERAGE.
    */
   viPrintf(vi, ":ACQUIRE:TYPE NORMAL\n");

   /* AQUIRE_COMPLETE - Specifies the minimum completion criteria
    * for an acquisition.  The parameter determines the percentage
    * of time buckets needed to be "full" before an acquisition is
    * considered to be complete.
    */
   viPrintf(vi, ":ACQUIRE:COMPLETE 100\n");

   /* DIGITIZE - Used to acquire the waveform data for transfer over
    * the interface.  Sending this command causes an acquisition to
    * take place with the resulting data being placed in the buffer.
    */

   /* NOTE!  The use of the DIGITIZE command is highly recommended
    * as it will ensure that sufficient data is available for
    * measurement.  Keep in mind when the oscilloscope is running,
    * communication with the computer interrupts data acquisition.
    * Setting up the oscilloscope over the bus causes the data
    * buffers to be cleared and internal hardware to be reconfigured.
    * If a measurement is immediately requested there may not have
    * been enough time for the data acquisition process to collect
    * data and the results may not be accurate.  An error value of
    * 9.9E+37 may be returned over the bus in this situation.
    */
   viPrintf(vi, ":DIGITIZE CHAN1\n");
}

/*
 * analyze
 * --------------------------------------------------------------------
 * In this example we will do the following:
 *  - Save the system setup to a file for restoration at a later time.
 *  - Save the oscilloscope display to a file which can be printed.
 *  - Make single channel measurements.
 */

void analyze (void)
{
   double frequency, vpp;           /* Measurements. */
   double vdiv, off, sdiv, delay;   /* Values calculated from preamble data. */
   int i;                           /* Loop counter. */
   unsigned char setup_string[SETUP_STR_SIZE];   /* Array for setup string. */
   int setup_size;
   FILE *fp;
   unsigned char image_data[IMG_SIZE];           /* Array for image data. */
   int img_size;

   /* SAVE_SYSTEM_SETUP - The :SYSTEM:SETUP? query returns a program message
    * that contains the current state of the instrument.  Its format is a
    * definite-length binary block, for example, #800002204<setup string><NL>
    * where the setup string is 2204 bytes in length.
    */
   setup_size = SETUP_STR_SIZE;
   /* Query and read setup string. */
   viQueryf(vi, ":SYSTEM:SETUP?\n", "%#b\n", &setup_size, setup_string);
   printf("Read setup string query (%d bytes).\n", setup_size);
   /* Write setup string to file. */
   fp = fopen ("c:\\scope\\config\\setup.dat", "wb");
   setup_size = fwrite(setup_string, sizeof(unsigned char), setup_size, fp);
   fclose (fp);
   printf("Wrote setup string (%d bytes) to file.\n", setup_size);

   /* RESTORE_SYSTEM_SETUP - Uploads a previously saved setup string to the
    * oscilloscope.
    */
   /* Read setup string from file. */
   fp = fopen ("c:\\scope\\config\\setup.dat", "rb");
   setup_size = fread (setup_string, sizeof(unsigned char), SETUP_STR_SIZE, fp);
   fclose (fp);
   printf("Read setup string (%d bytes) from file.\n", setup_size);
   /* Restore setup string. */
   viPrintf(vi, ":SYSTEM:SETUP #8%08d", setup_size);
   viBufWrite(vi, setup_string, setup_size, &setup_size);
   viPrintf(vi, "\n");
   printf("Restored setup string (%d bytes).\n", setup_size);

   /* IMAGE_TRANSFER - In this example we will query for the image
    * data with ":DISPLAY:DATA?" to read the data and save the data
    * to the file "image.dat" which you can then send to a printer.
    */
   viSetAttribute(vi, VI_ATTR_TMO_VALUE, 30000);
   printf("Transferring image to c:\\scope\\data\\screen.bmp\n");
   img_size = IMG_SIZE;
   viQueryf(vi, ":DISPLAY:DATA? BMP8bit, SCREEN, COLOR\n", "%#b\n", &img_size, image_data);
    printf("Read display data query (%d bytes).\n", img_size);
   /* Write image data to file. */
   fp = fopen ("c:\\scope\\data\\screen.bmp", "wb");
   img_size = fwrite(image_data, sizeof(unsigned char), img_size, fp);
   fclose (fp);
   printf("Wrote image data (%d bytes) to file.\n", img_size);
   viSetAttribute(vi, VI_ATTR_TMO_VALUE, 5000);

   /* MEASURE - The commands in the MEASURE subsystem are used to
    * make measurements on displayed waveforms.
    */
   viPrintf(vi, ":MEASURE:SOURCE CHANNEL1\n");   /* Set source to measure. */

   /* Query for frequency. */
   viQueryf(vi, ":MEASURE:FREQUENCY?\n", "%lf", &frequency);
   printf("The frequency is: %.4f kHz\n", frequency / 1000);
   /* Query for peak to peak voltage. */
   viQueryf(vi, ":MEASURE:VPP?\n", "%lf", &vpp);
   printf("The peak to peak voltage is: %.2f V\n", vpp);

   /* WAVEFORM_DATA - Get waveform data from oscilloscope.
    */
   get_waveform();

   /* Make some calculations from the preamble data. */
   vdiv  = 32 * preamble [7];
   off   = preamble [8];
   sdiv  = preamble [2] * preamble [4] / 10;
   delay = (preamble [2] / 2) * preamble [4] + preamble [5];

   /* Print them out... */
   printf ("Scope Settings for Channel 1:\n");
   printf ("Volts per Division = %f\n", vdiv);
   printf ("Offset = %f\n", off);
   printf ("Seconds per Division = %f\n", sdiv);
   printf ("Delay = %f\n", delay);

   /* print out the waveform voltage at selected points */
   for (i = 0; i < 1000; i = i + 50)
      printf ("Data Point %4d = %6.2f Volts at %10f Seconds\n", i,
      ((float)waveform_data[i] - preamble[9]) * preamble[7] + preamble[8],
      ((float)i - preamble[6]) * preamble[4] + preamble[5]);

   save_waveform();       /* Save waveform data to disk. */
   retrieve_waveform();   /* Load waveform data from disk. */
}

/*
 * get_waveform
 * ------------------------------------------------------------------
 * This function transfers the data displayed on the oscilloscope to
 * the computer for storage, plotting, or further analysis.
 */

void get_waveform (void)
{
   int waveform_size;

   /* WAVEFORM_DATA - To obtain waveform data, you must specify the
    * WAVEFORM parameters for the waveform data prior to sending the
    * ":WAVEFORM:DATA?" query.
    *
    * Once these parameters have been sent, the ":WAVEFORM:PREAMBLE?"
    * query provides information concerning the vertical and horizontal
    * scaling of the waveform data.
    *
    * With the preamble information you can then use the ":WAVEFORM:DATA?"
    * query and read the data block in the correct format.
    */

   /* WAVE_FORMAT - Sets the data transmission mode for waveform data
    * output.  This command controls how the data is formatted when
    * sent from the oscilloscope and can be set to WORD or BYTE format.
    */
   viPrintf(vi, ":WAVEFORM:FORMAT BYTE\n");   /* Set waveform format to BYTE. */

   /* WAVE_POINTS - Sets the number of points to be transferred.  The
    * number of time points available is returned by the "ACQUIRE:POINTS?"
    * query.  This can be set to any binary fraction of the total time
    * points available.
    */
   viPrintf(vi, ":WAVEFORM:POINTS 1000\n");
   /* GET_PREAMBLE - The preamble contains all of the current WAVEFORM
    * settings returned in the form <preamble block><NL> where the
    * <preamble block> is:
    *    FORMAT     : int16 - 0 = BYTE, 1 = WORD, 2 = ASCII.
    *    TYPE       : int16 - 0 = NORMAL, 1 = PEAK DETECT, 2 = AVERAGE.
    *    POINTS     : int32 - number of data points transferred.
    *    COUNT      : int32 - 1 and is always 1.
    *    XINCREMENT : float64 - time difference between data points.
    *    XORIGIN    : float64 - always the first data point in memory.
    *    XREFERENCE : int32 - specifies the data point associated w/x-origin.
    *    YINCREMENT : float32 - voltage difference between data points.
    *    YORIGIN    : float32 - value of the voltage at center screen.
    *    YREFERENCE : int32 - data point where y-origin occurs.
    */
   printf("Reading preamble\n");
   viQueryf(vi, ":WAVEFORM:PREAMBLE?\n", "%,10lf\n", preamble);
   /*
   printf("Preamble FORMAT: %e\n", preamble[0]);
   printf("Preamble TYPE: %e\n", preamble[1]);
   printf("Preamble POINTS: %e\n", preamble[2]);
   printf("Preamble COUNT: %e\n", preamble[3]);
   printf("Preamble XINCREMENT: %e\n", preamble[4]);
   printf("Preamble XORIGIN: %e\n", preamble[5]);
   printf("Preamble XREFERENCE: %e\n", preamble[6]);
   printf("Preamble YINCREMENT: %e\n", preamble[7]);
   printf("Preamble YORIGIN: %e\n", preamble[8]);
   printf("Preamble YREFERENCE: %e\n", preamble[9]);
   */

   /* QUERY_WAVE_DATA - Outputs waveform records to the controller
    * over the interface that is stored in a buffer previously
    * specified with the ":WAVEFORM:SOURCE" command.
    */
   viPrintf(vi, ":WAVEFORM:DATA?\n");   /* Query waveform data. */

   /* READ_WAVE_DATA - The wave data consists of two parts: the header,
    * and the actual waveform data followed by an New Line (NL) character.
    * The query data has the following format:
    *
    *    <header><waveform data block><NL>
    *
    * Where:
    *
    *    <header> = #800002048    (this is an example header)
    *
    * The "#8" may be stripped off of the header and the remaining
    * numbers are the size, in bytes, of the waveform data block.
    * The size can vary depending on the number of points acquired
    * for the waveform which can be set using the ":WAVEFORM:POINTS"
    * command.  You may then read that number of bytes from the
    * oscilloscope; then, read the following NL character to
    * terminate the query.
    */
   waveform_size = WAVE_DATA_SIZE;
   /* Read waveform data. */
   viScanf(vi, "%#b\n", &waveform_size, waveform_data);
   if ( waveform_size == WAVE_DATA_SIZE )
   {
      printf("Waveform data buffer full: May not have received all points.\n");
   }
   else
   {
      printf("Reading waveform data... size = %d\n", waveform_size);
   }
}
```

```
/*
 * save_waveform
 * ------------------------------------------------------------------
 * This function saves the waveform data from the get_waveform
 * function to disk.  The data is saved to a file called "wave.dat".
 */

void save_waveform(void)
{
   FILE *fp;

   fp = fopen("c:\\scope\\data\\wave.dat", "wb");
   fwrite(preamble, sizeof(preamble[0]), 10, fp);   /* Write preamble. */
   /* Write actually waveform data. */
   fwrite(waveform_data, sizeof(waveform_data[0]), (int)preamble[2], fp);
   fclose(fp);
}

/*
 * retrieve_waveform
 * ------------------------------------------------------------------
 * This function retrieves previously saved waveform data from a
 * file called "wave.dat".
 */

void retrieve_waveform(void)
{
   FILE *fp;

   fp = fopen("c:\\scope\\data\\wave.dat", "rb");
   fread(preamble, sizeof(preamble[0]), 10, fp);   /* Read preamble. */
   /* Read the waveform data. */
   fread(waveform_data, sizeof(waveform_data[0]), (int)preamble[2], fp);
   fclose(fp);
}
```

Programming Examples

VISA Example in Visual Basic

```
'
' Agilent VISA Example in Visual Basic
' -------------------------------------------------------------------
' This program illustrates most of the commonly-used programming
' features of the Agilent DSO/MSO6000-series oscilloscopes.
' -------------------------------------------------------------------

Option Explicit

Public err As Long   ' Error returned by VISA function calls.
Public drm As Long   ' Session to Default Resource Manager.
Public vi As Long    ' Session to instrument.

' Declare variables to hold numeric values returned by viVScanf/viVQueryf.
Public dblQueryResult As Double
Public Const DblArraySize = 20
Public Const ByteArraySize = 5000000
Public retCount As Long
Public dblArray(DblArraySize) As Double
Public byteArray(ByteArraySize) As Byte
Public paramsArray(2) As Long

' Declare fixed length string variable to hold string value returned
' by viVScanf/viVQueryf.
```

```
Public strQueryResult As String * 200

'
' MAIN PROGRAM
' -------------------------------------------------------------------
' This example shows the fundamental parts of a program (initialize,
' capture, analyze).
'
' The commands sent to the oscilloscope are written in both long and
' short form.  Both forms are acceptable.
'
' The input signal is the probe compensation signal from the front
' panel of the oscilloscope connected to channel 1.
'
' If you are using a different signal or different channels, these
' commands may not work as explained in the comments.
' -------------------------------------------------------------------

Sub Main()

  ' Open the default resource manager session.
  err = viOpenDefaultRM(drm)

  '  Open the session to the resource.
  '  The "GPIB0" parameter is the VISA Interface name to
  '  an GPIB instrument as defined in:
  '    Start->Programs->Agilent IO Libraries->IO Config
  '  Change this name to whatever you have defined for your
  '  VISA Interface.
  '  "GPIB0::7::INSTR" is the address string for the device -
  '  this address will be the same as seen in:
  '    Start->Programs->Agilent IO Libraries->VISA Assistant
  '  (after the VISA Interface Name is defined in IO Config).

  ' err = viOpen(drm, "GPIB0::7::INSTR", 0, 0, vi)
  ' err = viOpen(drm, "TCPIP0::a-mso6102-90541::inst0::INSTR", 0, 0, vi)
  err = viOpen(drm, "USB0::2391::5970::30D3090541::0::INSTR", 0, 60000, vi)

  ' Initialize - Initialization will start the program with the
  ' oscilloscope in a known state.
  Initialize

  ' Capture - After initialization, you must make waveform data
  ' available to analyze.  To do this, capture the data using the
  ' DIGITIZE command.
  Capture

  ' Analyze - Once the waveform has been captured, it can be analyzed.
  ' There are many parts of a waveform to analyze.  This example shows
  ' some of the possible ways to analyze various parts of a waveform.
  Analyze

  ' Close the vi session and the resource manager session.
  err = viClose(vi)
  err = viClose(drm)

End Sub

'
' Initialize
' -------------------------------------------------------------------
' Initialize will start the program with the oscilloscope in a known
' state.  This is required because some uninitialized conditions could
' cause the program to fail or not perform as expected.
'
' In this example, we initialize the following:
'  - Oscilloscope
'  - Channel 1 range
'  - Display Grid
'  - Timebase reference, range, and delay
'  - Trigger mode and type
'
' There are also some additional initialization commands, which are
' not used, but shown for reference.
' -------------------------------------------------------------------

Private Sub Initialize()

  ' Clear the interface.
  err = viClear(vi)

  ' RESET - This command puts the oscilloscope into a known state.
  ' This statement is very important for programs to work as expected.
  ' Most of the following initialization commands are initialized by
  ' *RST.  It is not necessary to reinitialize them unless the default
  ' setting is not suitable for your application.
  err = viVPrintf(vi, "*RST" + vbLf, 0)    ' Reset the oscilloscope to the defaults.

  ' IDN - Ask for the device's *IDN string.
  err = viVPrintf(vi, "*IDN?" + vbLf, 0)
  err = viVScanf(vi, "%t", strQueryResult)  ' Read the results as a string.
  MsgBox "Result is: " + strQueryResult, vbOKOnly, "*IDN? Result"   ' Display results.

  ' AUTOSCALE - This command evaluates all the input signals and sets
  ' the correct conditions to display all of the active signals.
  err = viVPrintf(vi, ":AUTOSCALE" + vbLf, 0)    ' Same as pressing the Autoscale key.

  ' CHANNEL_PROBE - Sets the probe attenuation factor for the selected
  ' channel.  The probe attenuation factor may be set from 0.1 to 1000.
  err = viVPrintf(vi, ":CHAN1:PROBE 10" + vbLf, 0)    ' Set Probe to 10:1.

  ' CHANNEL_RANGE - Sets the full scale vertical range in volts.  The
  ' range value is 8 times the volts per division.
  err = viVPrintf(vi, ":CHANNEL1:RANGE 8" + vbLf, 0)    ' Set the vertical range to 8 volts.

  ' TIME_RANGE - Sets the full scale horizontal time in seconds.  The
  ' range value is 10 times the time per division.
  err = viVPrintf(vi, ":TIM:RANG 2e-3" + vbLf, 0)    ' Set the time range to 0.002 seconds.

  ' TIME_REFERENCE - Possible values are LEFT and CENTER.
  '  - LEFT sets the display reference on time division from the left.
  '  - CENTER sets the display reference to the center of the screen.
  err = viVPrintf(vi, ":TIMEBASE:REFERENCE CENTER" + vbLf, 0)    ' Set reference to center.

  ' TRIGGER_TV_SOURCE - Selects the channel that actuall produces the
  ' TV trigger.  Any channel can be selected.
  err = viVPrintf(vi, ":TRIGGER:TV:SOURCE CHANNEL1" + vbLf, 0)

  ' TRIGGER_MODE - Set the trigger mode to EDGE, GLITch, PATTern, CAN,
  ' DURation, IIC, LIN, SEQuence, SPI, TV, or USB.
  err = viVPrintf(vi, ":TRIGGER:MODE EDGE" + vbLf, 0)   ' Set the trigger mode to EDGE.

  ' TRIGGER_EDGE_SLOPE - Sets the slope of the edge for the trigger.
  err = viVPrintf(vi, ":TRIGGER:EDGE:SLOPE POSITIVE" + vbLf, 0)   ' Set the slope to positive

  ' The following commands are not executed and are shown for reference
  ' purposes only.  To execute these commands, uncomment them.

  ' RUN_STOP - (not executed in this example)
  '  - RUN starts the acquisition of data for the active waveform display.
  '  - STOP stops the data acquisition and turns off AUTOSTORE.
  ' err = viVPrintf(vi, ":RUN" + vbLf, 0)   ' Start data acquisition.
  ' err = viVPrintf(vi, ":STOP" + vbLf, 0)   ' Stop the data acquisition.
  ' VIEW_BLANK - (not executed in this example)
  '  - VIEW turns on (starts displaying) a channel or pixel memory.
  '  - BLANK turns off (stops displaying) a channel or pixel memory.
  ' err = viVPrintf(vi, ":BLANK CHANNEL1" + vbLf, 0)   ' Turn channel 1 off.
  ' err = viVPrintf(vi, ":VIEW CHANNEL1" + vbLf, 0)   ' Turn channel 1 on.

  ' TIMEBASE_MODE - (not executed in this example)
  ' Set the time base mode to MAIN, DELAYED, XY, or ROLL.
  ' err = viVPrintf(vi, ":TIMEBASE:MODE MAIN" + vbLf, 0)   ' Set time base mode to main.

End Sub

'
' Capture
' --------------------------------------------------------------------
' We will capture the waveform using the digitize command.
' --------------------------------------------------------------------

Private Sub Capture()

  ' AQUIRE_TYPE - Sets the acquisition mode, which can be NORMAL,
  ' PEAK, or AVERAGE.
  err = viVPrintf(vi, ":ACQUIRE:TYPE NORMAL" + vbLf, 0)

  ' AQUIRE_COMPLETE - Specifies the minimum completion criteria for
  ' an acquisition.  The parameter determines the percentage of time
  ' buckets needed to be "full" before an acquisition is considered
  ' to be complete.
  err = viVPrintf(vi, ":ACQUIRE:COMPLETE 100" + vbLf, 0)

  ' DIGITIZE - Used to acquire the waveform data for transfer over
  ' the interface.  Sending this command causes an acquisition to
  ' take place with the resulting data being placed in the buffer.
  '
  ' NOTE!  The DIGITIZE command is highly recommended for triggering
  ' modes other than SINGLE.  This ensures that sufficient data is
  ' available for measurement.  If DIGITIZE is used with single mode,
  ' the completion criteria may never be met.  The number of points
  ' gathered in Single mode is related to the sweep speed, memory
  ' depth, and maximum sample rate.  For example, take an oscilloscope
  ' with a 1000-point memory, a sweep speed of 10 us/div (100 us
  ' total time across the screen), and a 20 MSa/s maximum sample rate.
  ' 1000 divided by 100 us equals 10 MSa/s.  Because this number is
  ' less than or equal to the maximum sample rate, the full 1000 points
  ' will be digitized in a single acquisition.  Now, use 1 us/div
  ' (10 us across the screen).  1000 divided by 10 us equals 100 MSa/s;
  ' because this is greater than the maximum sample rate by 5 times,
  ' only 400 points (or 1/5 the points) can be gathered on a single
  ' trigger.  Keep in mind when the oscilloscope is running, communication
  ' with the computer interrupts data acquisition.  Setting up the
  ' oscilloscope over the bus causes the data buffers to be cleared
  ' and internal hardware to be reconfigured.  If a measurement is
  ' immediately requested, there may have not been enough time for
  ' the data acquisition process to collect data, and the results may
  ' not be accurate.  An error value of 9.9E+37 may be returned over
  ' the bus in this situation.
  '
  err = viVPrintf(vi, ":DIGITIZE CHAN1" + vbLf, 0)

End Sub

'
' Analyze
' --------------------------------------------------------------------
' In analyze, we will do the following:
'  - Save the system setup to a file and restore it.
'  - Save the waveform data to a file on the computer.
'  - Make single channel measurements.
'  - Save the oscilloscope display to a file that can be sent to a printer.
' ------------------------------------------------------------------

Private Sub Analyze()

  ' Set up arrays for multiple parameter query returning an array
  ' with viVScanf/viVQueryf.  Set retCount to the maximum number
  ' of elements that the array can hold.
  paramsArray(0) = VarPtr(retCount)
  paramsArray(1) = VarPtr(byteArray(0))

  ' SAVE_SYSTEM_SETUP - The :SYSTEM:SETUP? query returns a program message
  ' that contains the current state of the instrument.  Its format is a
  ' definite-length binary block, for example, #800002204<setup string><NL>
  ' where the setup string is 2204 bytes in length.
  Dim lngSetupStringSize As Long
  err = viVPrintf(vi, ":SYSTEM:SETUP?" + vbLf, 0)
  retCount = ByteArraySize
  err = viVScanf(vi, "%#b\n" + vbLf, paramsArray(0))   ' Unsigned integer bytes.
  lngSetupStringSize = retCount
  ' Output setup string to a file:
  Dim strPath As String
  Dim lngI As Long
  strPath = "c:\scope\config\setup.dat"
  Close #1   ' If #1 is open, close it.
  Open strPath For Binary Access Write Lock Write As #1   ' Open file for output.
  For lngI = 0 To lngSetupStringSize - 1
    Put #1, , byteArray(lngI)   ' Write data.
  Next lngI
  Close #1   ' Close file.

  ' IMAGE_TRANSFER - In this example, we will query for the image data
  ' with ":DISPLAY:DATA?", read the data, and then save it to a file.
  err = viVPrintf(vi, ":DISPLAY:DATA? BMP, SCREEN, COLOR" + vbLf, 0)
  retCount = ByteArraySize
  err = viVScanf(vi, "%#b\n" + vbLf, paramsArray(0))   ' Unsigned integer bytes.
  ' Output display data to a file:
  strPath = "c:\scope\data\screen.bmp"
  ' Remove file if it exists.
  If Len(Dir(strPath)) Then
    Kill strPath
  End If
  Close #1   ' If #1 is open, close it.
  Open strPath For Binary Access Write Lock Write As #1   ' Open file for output.
  For lngI = 0 To retCount - 1
    Put #1, , byteArray(lngI)   ' Write data.
  Next lngI
  Close #1   ' Close file.

  ' RESTORE_SYSTEM_SETUP - Read the setup string from a file and write it
  ' back to the oscilloscope.
  strPath = "c:\scope\config\setup.dat"
  Open strPath For Binary Access Read As #1   ' Open file for input.
  Get #1, , byteArray   ' Read data.
  Close #1   ' Close file.
  ' Write learn string back to oscilloscope using ":SYSTEM:SETUP" command:
  retCount = lngSetupStringSize
  err = viVPrintf(vi, ":SYSTEM:SETUP %#b" + vbLf, paramsArray(0))

  ' MEASURE - The commands in the MEASURE subsystem are used to make
  ' measurements on displayed waveforms.
  err = viVPrintf(vi, ":MEASURE:SOURCE CHANNEL1" + vbLf, 0)   ' Source to measure

  err = viVPrintf(vi, ":MEASURE:FREQUENCY?" + vbLf, 0)   ' Query for frequency.
  err = viVScanf(vi, "%lf" + vbLf, VarPtr(dblQueryResult))   ' Read frequency.
  MsgBox "Frequency:" + vbCrLf + FormatNumber(dblQueryResult / 1000, 4) + " kHz"
```

```
err = viVPrintf(vi, ":MEASURE:DUTYCYCLE?" + vbLf, 0)   ' Query for duty cycle.
err = viVScanf(vi, "%lf" + vbLf, VarPtr(dblQueryResult))   ' Read duty cycle.
MsgBox "Duty cycle:" + vbCrLf + FormatNumber(dblQueryResult, 3) + "%"

err = viVPrintf(vi, ":MEASURE:RISETIME?" + vbLf, 0)   ' Query for risetime.
err = viVScanf(vi, "%lf" + vbLf, VarPtr(dblQueryResult))   ' Read risetime.
MsgBox "Risetime:" + vbCrLf + FormatNumber(dblQueryResult * 1000000, 4) + " us"

err = viVPrintf(vi, ":MEASURE:VPP?" + vbLf, 0)   ' Query for Peak to Peak voltage.
err = viVScanf(vi, "%lf" + vbLf, VarPtr(dblQueryResult))   ' Read VPP.
MsgBox "Peak to peak voltage:" + vbCrLf + FormatNumber(dblQueryResult, 4) + " V"

err = viVPrintf(vi, ":MEASURE:VMAX?" + vbLf, 0)   ' Query for Vmax.
err = viVScanf(vi, "%lf" + vbLf, VarPtr(dblQueryResult))   ' Read Vmax.
MsgBox "Maximum voltage:" + vbCrLf + FormatNumber(dblQueryResult, 4) + " V"

' WAVEFORM_DATA - To obtain waveform data, you must specify the
' WAVEFORM parameters for the waveform data prior to sending the
' ":WAVEFORM:DATA?" query.  Once these parameters have been sent,
' the waveform data and the preamble can be read.
'
' WAVE_SOURCE - Selects the channel to be used as the source for
' the waveform commands.
err = viVPrintf(vi, ":WAVEFORM:SOURCE CHAN1" + vbLf, 0)

' WAVE_POINTS - Specifies the number of points to be transferred
' using the ":WAVEFORM:DATA?" query.
err = viVPrintf(vi, ":WAVEFORM:POINTS 1000" + vbLf, 0)

' WAVE_FORMAT - Sets the data transmission mode for the waveform
' data output.  This command controls whether data is formatted in
' a word or byte format when sent from the oscilloscope.
Dim lngVSteps As Long
Dim intBytesPerData As Integer

err = viVPrintf(vi, ":WAVEFORM:FORMAT WORD" + vbLf, 0)   ' Data in range 0 to 65535.
lngVSteps = 65536
intBytesPerData = 2
'err = viVPrintf(vi, ":WAVEFORM:FORMAT BYTE" + vbLf, 0)   ' Data in range 0 to 255.
'lngVSteps = 256
'intBytesPerData = 1

' GET_PREAMBLE - The preamble block contains all of the current
' WAVEFORM settings.  It is returned in the form <preamble_block><NL>
' where <preamble_block> is:
'    FORMAT        : int16 - 0 = BYTE, 1 = WORD, 2 = ASCII.
'    TYPE          : int16 - 0 = NORMAL, 1 = PEAK DETECT, 2 = AVERAGE.
'    POINTS        : int32 - number of data points transferred.
'    COUNT         : int32 - 1 and is always 1.
'    XINCREMENT    : float64 - time difference between data points.
'    XORIGIN       : float64 - always the first data point in memory.
'    XREFERENCE    : int32 - specifies the data point associated with x-origin.
'    YINCREMENT    : float32 - voltage difference between data points.
'    YORIGIN       : float32 - value is the voltage at center screen.
'    YREFERENCE    : int32 - specifies the data point where y-origin occurs.
Dim intFormat As Integer
Dim intType As Integer
Dim lngPoints As Long
Dim lngCount As Long
Dim dblXIncrement As Double
Dim dblXOrigin As Double
Dim lngXReference As Long
Dim sngYIncrement As Single
Dim sngYOrigin As Single
Dim lngYReference As Long
Dim strOutput As String
```

```
err = viVPrintf(vi, ":WAVEFORM:PREAMBLE?" + vbLf, 0)   ' Query for the preamble.
paramsArray(1) = VarPtr(dblArray(0))
retCount = DblArraySize
err = viVScanf(vi, "%,#lf" + vbLf, paramsArray(0))   ' Read preamble information.
intFormat = dblArray(0)
intType = dblArray(1)
lngPoints = dblArray(2)
lngCount = dblArray(3)
dblXIncrement = dblArray(4)
dblXOrigin = dblArray(5)
lngXReference = dblArray(6)
sngYIncrement = dblArray(7)
sngYOrigin = dblArray(8)
lngYReference = dblArray(9)
strOutput = ""
'strOutput = strOutput + "Format = " + CStr(intFormat) + vbCrLf
'strOutput = strOutput + "Type = " + CStr(intType) + vbCrLf
'strOutput = strOutput + "Points = " + CStr(lngPoints) + vbCrLf
'strOutput = strOutput + "Count = " + CStr(lngCount) + vbCrLf
'strOutput = strOutput + "X increment = " + FormatNumber(dblXIncrement * 1000000) + _
'            " us" + vbCrLf
'strOutput = strOutput + "X origin = " + FormatNumber(dblXOrigin * 1000000) + _
'            " us" + vbCrLf
'strOutput = strOutput + "X reference = " + CStr(lngXReference) + vbCrLf
'strOutput = strOutput + "Y increment = " + FormatNumber(sngYIncrement * 1000) + _
'            " mV" + vbCrLf
'strOutput = strOutput + "Y origin = " + FormatNumber(sngYOrigin) + " V" + vbCrLf
'strOutput = strOutput + "Y reference = " + CStr(lngYReference) + vbCrLf
strOutput = strOutput + "Volts/Div = " + FormatNumber(lngVSteps * sngYIncrement / 8) + _
            " V" + vbCrLf
strOutput = strOutput + "Offset = " + FormatNumber(sngYOrigin) + " V" + vbCrLf
strOutput = strOutput + "Sec/Div = " + FormatNumber(lngPoints * dblXIncrement / 10 * _
            1000000) + " us" + vbCrLf
strOutput = strOutput + "Delay = " + FormatNumber(((lngPoints / 2) * _
            dblXIncrement + dblXOrigin) * 1000000) + " us" + vbCrLf

' QUERY_WAVE_DATA - Outputs waveform data that is stored in a buffer.
err = viVPrintf(vi, ":WAV:DATA?" + vbLf, 0)   ' Query the oscilloscope for the waveform dat

' READ_WAVE_DATA - The wave data consists of two parts: the header,
' and the actual waveform data followed by a new line (NL) character.
' The query data has the following format:
'
'    <header><waveform_data><NL>
'
' Where:
'
'    <header> = #800001000 (This is an example header)
'
' The "#8" may be stripped off of the header and the remaining
' numbers are the size, in bytes, of the waveform data block.  The
' size can vary depending on the number of points acquired for the
' waveform.  You can then read that number of bytes from the
' oscilloscope and the terminating NL character.
'
'Dim lngI As Long
Dim lngDataValue As Long

paramsArray(1) = VarPtr(byteArray(0))
retCount = ByteArraySize
err = viVScanf(vi, "%#b" + vbLf, paramsArray(0)) ' Unsigned integer bytes.
' retCount is now actual number of bytes returned by query.
For lngI = 0 To retCount - 1 Step (retCount / 20)   ' 20 points.
  If intBytesPerData = 2 Then
    lngDataValue = CLng(byteArray(lngI)) * 256 + CLng(byteArray(lngI + 1))   ' 16-bit value
  Else
```

```
      lngDataValue = CLng(byteArray(lngI))   ' 8-bit value.
    End If
    strOutput = strOutput + "Data point " + CStr(lngI / intBytesPerData) + ", " + _
      FormatNumber((lngDataValue - lngYReference) * sngYIncrement + sngYOrigin) + _
      " V, " + _
      FormatNumber(((lngI / intBytesPerData - lngXReference) * dblXIncrement + _
      dblXOrigin) * 1000000) + " us" + vbCrLf
  Next lngI
  MsgBox "Waveform data:" + vbCrLf + strOutput

  ' Make a delay measurement between channel 1 and 2.
  Dim dblChan1Edge1 As Double
  Dim dblChan2Edge1 As Double
  Dim dblChan1Edge2 As Double
  Dim dblDelay As Double
  Dim dblPeriod As Double
  Dim dblPhase As Double

  err = viVPrintf(vi, ":MEASURE:TEDGE? +1, CHAN1" + vbLf, 0)   ' Query time at 1st rising edg
  err = viVScanf(vi, "%lf", VarPtr(dblChan1Edge1))   ' Read time at edge 1 on ch 1.
  err = viVPrintf(vi, ":MEASURE:TEDGE? +1, CHAN2" + vbLf, 0)   ' Query time at 1st rising edg
  err = viVScanf(vi, "%lf", VarPtr(dblChan2Edge1))   ' Read time at edge 1 on ch 2.
  dblDelay = dblChan2Edge1 - dblChan1Edge1   ' Calculate delay time between ch1 and ch2.
  MsgBox "Delay = " + vbCrLf + CStr(dblDelay)   ' Write calculated delay time to screen.

  ' Make a phase difference measurement between channel 1 and 2.
  err = viVPrintf(vi, ":MEASURE:TEDGE? +2, CHAN1" + vbLf, 0)   ' Query time at 1st rising edg
  err = viVScanf(vi, "%lf", VarPtr(dblChan1Edge2))   ' Read time at edge 2 on ch 1.
  dblPeriod = dblChan1Edge2 - dblChan1Edge1   ' Calculate period of ch 1.
  dblPhase = (dblDelay / dblPeriod) * 360 ' Calculate phase difference between ch1 and ch2.
  MsgBox "Phase = " + vbCrLf + CStr(dblPhase)

End Sub
```

Programming Examples

VISA COM Example in Visual Basic

```
'
' Agilent VISA COM Example in Visual Basic
' -------------------------------------------------------------------
' This program illustrates most of the commonly used programming
' features of the Agilent DSO/MSO6000-series oscilloscopes.
' -------------------------------------------------------------------

Option Explicit

Public myMgr As VisaComLib.ResourceManager
Public myScope As VisaComLib.FormattedIO488
Public varQueryResult As Variant
Public strQueryResult As String

'
' MAIN PROGRAM
' -------------------------------------------------------------------
' This example shows the fundamental parts of a program (initialize,
' capture, analyze).
'
' The commands sent to the oscilloscope are written in both long and
' short form.  Both forms are acceptable.
'
' The input signal is the probe compensation signal from the front
' panel of the oscilloscope connected to channel 1.
'
```

```
' If you are using a different signal or different channels, these
' commands may not work as explained in the comments.
' -------------------------------------------------------------------

Sub Main()

  On Error GoTo VisaComError

  ' Create the VISA COM I/O resource.
  Set myMgr = New VisaComLib.ResourceManager
  Set myScope = New VisaComLib.FormattedIO488
  'Set myScope.IO = myMgr.Open("GPIB0::7::INSTR")   ' GPIB.
  'Set myScope.IO = myMgr.Open("TCPIP0::a-mso6102-90541::inst0::INSTR")   ' LAN.
  Set myScope.IO = myMgr.Open("USB0::2391::5970::30D3090541::0::INSTR")   ' USB.

  ' Initialize - Initialization will start the program with the
  ' oscilloscope in a known state.
  Initialize

  ' Capture - After initialization, you must make waveform data
  ' available to analyze.  To do this, capture the data using the
  ' DIGITIZE command.
  Capture

  ' Analyze - Once the waveform has been captured, it can be analyzed.
  ' There are many parts of a waveform to analyze.  This example shows
  ' some of the possible ways to analyze various parts of a waveform.
  Analyze

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error:" + vbCrLf + Err.Description

End Sub

'
' Initialize
' -------------------------------------------------------------------
' Initialize will start the program with the oscilloscope in a known
' state.  This is required because some uninitialized conditions could
' cause the program to fail or not perform as expected.
'
' In this example, we initialize the following:
'  - Oscilloscope
'  - Channel 1 range
'  - Display Grid
'  - Timebase reference, range, and delay
'  - Trigger mode and type
'
' There are also some additional initialization commands, which are
' not used, but shown for reference.
' -------------------------------------------------------------------

Private Sub Initialize()

  On Error GoTo VisaComError

  ' Clear the interface.
  myScope.IO.Clear

  ' RESET - This command puts the oscilloscope into a known state.
  ' This statement is very important for programs to work as expected.
  ' Most of the following initialization commands are initialized by
  ' *RST.  It is not necessary to reinitialize them unless the default
  ' setting is not suitable for your application.
  myScope.WriteString "*RST"   ' Reset the oscilloscope to the defaults.
  ' AUTOSCALE - This command evaluates all the input signals and sets
  ' the correct conditions to display all of the active signals.
  myScope.WriteString ":AUTOSCALE"   ' Same as pressing the Autoscale key.

  ' CHANNEL_PROBE - Sets the probe attenuation factor for the selected
  ' channel.  The probe attenuation factor may be set from 0.1 to 1000.
  myScope.WriteString ":CHAN1:PROBE 10"   ' Set Probe to 10:1.

  ' CHANNEL_RANGE - Sets the full scale vertical range in volts.  The
  ' range value is 8 times the volts per division.
  myScope.WriteString ":CHANNEL1:RANGE 8"   ' Set the vertical range to 8 volts.

  ' TIME_RANGE - Sets the full scale horizontal time in seconds.  The
  ' range value is 10 times the time per division.
  myScope.WriteString ":TIM:RANG 2e-3"   ' Set the time range to 0.002 seconds.

  ' TIME_REFERENCE - Possible values are LEFT and CENTER.
  '  - LEFT sets the display reference on time division from the left.
  '  - CENTER sets the display reference to the center of the screen.
  myScope.WriteString ":TIMEBASE:REFERENCE CENTER"   ' Set reference to center.

  ' TRIGGER_TV_SOURCE - Selects the channel that actuall produces the
  ' TV trigger.  Any channel can be selected.
  myScope.WriteString ":TRIGGER:TV:SOURCE CHANNEL1"

  ' TRIGGER_MODE - Set the trigger mode to EDGE, GLITch, PATTern, CAN,
  ' DURation, IIC, LIN, SEQuence, SPI, TV, or USB.
  myScope.WriteString ":TRIGGER:MODE EDGE"   ' Set the trigger mode to EDGE.

  ' TRIGGER_EDGE_SLOPE - Sets the slope of the edge for the trigger.
  myScope.WriteString ":TRIGGER:EDGE:SLOPE POSITIVE"   ' Set the slope to positive.

  ' The following commands are not executed and are shown for reference
  ' purposes only.  To execute these commands, uncomment them.

  ' RUN_STOP - (not executed in this example)
  '  - RUN starts the acquisition of data for the active waveform display.
  '  - STOP stops the data acquisition and turns off AUTOSTORE.
  ' myScope.WriteString ":RUN"   ' Start data acquisition.
  ' myScope.WriteString ":STOP"   ' Stop the data acquisition.

  ' VIEW_BLANK - (not executed in this example)
  '  - VIEW turns on (starts displaying) a channel or pixel memory.
  '  - BLANK turns off (stops displaying) a channel or pixel memory.
  ' myScope.WriteString ":BLANK CHANNEL1"   ' Turn channel 1 off.
  ' myScope.WriteString ":VIEW CHANNEL1"   ' Turn channel 1 on.

  ' TIMEBASE_MODE - (not executed in this example)
  ' Set the time base mode to MAIN, DELAYED, XY, or ROLL.
  ' myScope.WriteString ":TIMEBASE:MODE MAIN"   ' Set time base mode to main.

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error:" + vbCrLf + Err.Description

End Sub

'
' Capture
' ---------------------------------------------------------------------
' We will capture the waveform using the digitize command.
' ---------------------------------------------------------------------

Private Sub Capture()
```

```
  On Error GoTo VisaComError

  ' AQUIRE_TYPE - Sets the acquisition mode, which can be NORMAL,
  ' PEAK, or AVERAGE.
  myScope.WriteString ":ACQUIRE:TYPE NORMAL"

  ' AQUIRE_COMPLETE - Specifies the minimum completion criteria for
  ' an acquisition.  The parameter determines the percentage of time
  ' buckets needed to be "full" before an acquisition is considered
  ' to be complete.
  myScope.WriteString ":ACQUIRE:COMPLETE 100"

  ' DIGITIZE - Used to acquire the waveform data for transfer over
  ' the interface.  Sending this command causes an acquisition to
  ' take place with the resulting data being placed in the buffer.
  '
  ' NOTE!  The DIGITIZE command is highly recommended for triggering
  ' modes other than SINGLE.  This ensures that sufficient data is
  ' available for measurement.  If DIGITIZE is used with single mode,
  ' the completion criteria may never be met.  The number of points
  ' gathered in Single mode is related to the sweep speed, memory
  ' depth, and maximum sample rate.  For example, take an oscilloscope
  ' with a 1000-point memory, a sweep speed of 10 us/div (100 us
  ' total time across the screen), and a 20 MSa/s maximum sample rate.
  ' 1000 divided by 100 us equals 10 MSa/s.  Because this number is
  ' less than or equal to the maximum sample rate, the full 1000 points
  ' will be digitized in a single acquisition.  Now, use 1 us/div
  ' (10 us across the screen).  1000 divided by 10 us equals 100 MSa/s;
  ' because this is greater than the maximum sample rate by 5 times,
  ' only 400 points (or 1/5 the points) can be gathered on a single
  ' trigger.  Keep in mind when the oscilloscope is running, communication
  ' with the computer interrupts data acquisition.  Setting up the
  ' oscilloscope over the bus causes the data buffers to be cleared
  ' and internal hardware to be reconfigured.  If a measurement is
  ' immediately requested, there may have not been enough time for
  ' the data acquisition process to collect data, and the results may
  ' not be accurate.  An error value of 9.9E+37 may be returned over
  ' the bus in this situation.
  '
  myScope.WriteString ":DIGITIZE CHAN1"

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error:" + vbCrLf + Err.Description

End Sub

'
' Analyze
' --------------------------------------------------------------
' In analyze, we will do the following:
'  - Save the system setup to a file and restore it.
'  - Save the waveform data to a file on the computer.
'  - Make single channel measurements.
'  - Save the oscilloscope display to a file that can be sent to a printer.
' --------------------------------------------------------------

Private Sub Analyze()

  On Error GoTo VisaComError

  ' SAVE_SYSTEM_SETUP - The :SYSTEM:SETUP? query returns a program message
  ' that contains the current state of the instrument.  Its format is a
  ' definite-length binary block, for example, #800002204<setup string><NL>
  ' where the setup string is 2204 bytes in length.
  myScope.WriteString ":SYSTEM:SETUP?"
```

```
varQueryResult = myScope.ReadIEEEBlock(BinaryType_UI1)
CheckForInstrumentErrors   ' After reading query results.
' Output setup string to a file:
Dim strPath As String
strPath = "c:\scope\config\setup.dat"
Close #1   ' If #1 is open, close it.
Open strPath For Binary Access Write Lock Write As #1   ' Open file for output.
Put #1, , varQueryResult   ' Write data.
Close #1   ' Close file.

' IMAGE_TRANSFER - In this example, we will query for the image data
' with ":DISPLAY:DATA?", read the data, and then save it to a file.
Dim byteData() As Byte
myScope.IO.Timeout = 15000
myScope.WriteString ":DISPLAY:DATA? BMP, SCREEN, COLOR"
byteData = myScope.ReadIEEEBlock(BinaryType_UI1)
' Output display data to a file:
strPath = "c:\scope\data\screen.bmp"
' Remove file if it exists.
If Len(Dir(strPath)) Then
  Kill strPath
End If
Close #1   ' If #1 is open, close it.
Open strPath For Binary Access Write Lock Write As #1   ' Open file for output.
Put #1, , byteData   ' Write data.
Close #1   ' Close file.
myScope.IO.Timeout = 5000

' RESTORE_SYSTEM_SETUP - Read the setup string from a file and write it
' back to the oscilloscope.
Dim varSetupString As Variant
strPath = "c:\scope\config\setup.dat"
Open strPath For Binary Access Read As #1   ' Open file for input.
Get #1, , varSetupString   ' Read data.
Close #1   ' Close file.
' Write setup string back to oscilloscope using ":SYSTEM:SETUP" command:
myScope.WriteIEEEBlock ":SYSTEM:SETUP ", varSetupString
CheckForInstrumentErrors

' MEASURE - The commands in the MEASURE subsystem are used to make
' measurements on displayed waveforms.
myScope.WriteString ":MEASURE:SOURCE CHANNEL1"   ' Source to measure
myScope.WriteString ":MEASURE:FREQUENCY?"   ' Query for frequency.
varQueryResult = myScope.ReadNumber   ' Read frequency.
MsgBox "Frequency:" + vbCrLf + FormatNumber(varQueryResult / 1000, 4) + " kHz"
myScope.WriteString ":MEASURE:DUTYCYCLE?"   ' Query for duty cycle.
varQueryResult = myScope.ReadNumber   ' Read duty cycle.
MsgBox "Duty cycle:" + vbCrLf + FormatNumber(varQueryResult, 3) + "%"
myScope.WriteString ":MEASURE:RISETIME?"   ' Query for risetime.
varQueryResult = myScope.ReadNumber   ' Read risetime.
MsgBox "Risetime:" + vbCrLf + FormatNumber(varQueryResult * 1000000, 4) + " us"
myScope.WriteString ":MEASURE:VPP?"   ' Query for Peak to Peak voltage.
varQueryResult = myScope.ReadNumber   ' Read VPP.
MsgBox "Peak to peak voltage:" + vbCrLf + FormatNumber(varQueryResult, 4) + " V"
myScope.WriteString ":MEASURE:VMAX?"   ' Query for Vmax.
varQueryResult = myScope.ReadNumber   ' Read Vmax.
MsgBox "Maximum voltage:" + vbCrLf + FormatNumber(varQueryResult, 4) + " V"

' WAVEFORM_DATA - To obtain waveform data, you must specify the
' WAVEFORM parameters for the waveform data prior to sending the
' ":WAVEFORM:DATA?" query.  Once these parameters have been sent,
' the waveform data and the preamble can be read.
'
' WAVE_SOURCE - Selects the channel to be used as the source for
' the waveform commands.
myScope.WriteString ":WAVEFORM:SOURCE CHAN1"
```

```
' WAVE_POINTS - Specifies the number of points to be transferred
' using the ":WAVEFORM:DATA?" query.
myScope.WriteString ":WAVEFORM:POINTS 1000"

' WAVE_FORMAT - Sets the data transmission mode for the waveform
' data output.  This command controls whether data is formatted in
' a word or byte format when sent from the oscilloscope.
Dim lngVSteps As Long
Dim intBytesPerData As Integer

myScope.WriteString ":WAVEFORM:FORMAT WORD"   ' Data in range 0 to 65535.
lngVSteps = 65536
intBytesPerData = 2
'myScope.WriteString ":WAVEFORM:FORMAT BYTE"   ' Data in range 0 to 255.
'lngVSteps = 256
'intBytesPerData = 1

' GET_PREAMBLE - The preamble block contains all of the current
' WAVEFORM settings.  It is returned in the form <preamble_block><NL>
' where <preamble_block> is:
'    FORMAT        : int16 - 0 = BYTE, 1 = WORD, 2 = ASCII.
'    TYPE          : int16 - 0 = NORMAL, 1 = PEAK DETECT, 2 = AVERAGE.
'    POINTS        : int32 - number of data points transferred.
'    COUNT         : int32 - 1 and is always 1.
'    XINCREMENT    : float64 - time difference between data points.
'    XORIGIN       : float64 - always the first data point in memory.
'    XREFERENCE    : int32 - specifies the data point associated with x-origin.
'    YINCREMENT    : float32 - voltage difference between data points.
'    YORIGIN       : float32 - value is the voltage at center screen.
'    YREFERENCE    : int32 - specifies the data point where y-origin occurs.
Dim Preamble()
Dim intFormat As Integer
Dim intType As Integer
Dim lngPoints As Long
Dim lngCount As Long
Dim dblXIncrement As Double
Dim dblXOrigin As Double
Dim lngXReference As Long
Dim sngYIncrement As Single
Dim sngYOrigin As Single
Dim lngYReference As Long
Dim strOutput As String

myScope.WriteString ":WAVEFORM:PREAMBLE?"   ' Query for the preamble.
Preamble() = myScope.ReadList   ' Read preamble information.
intFormat = Preamble(0)
intType = Preamble(1)
lngPoints = Preamble(2)
lngCount = Preamble(3)
dblXIncrement = Preamble(4)
dblXOrigin = Preamble(5)
lngXReference = Preamble(6)
sngYIncrement = Preamble(7)
sngYOrigin = Preamble(8)
lngYReference = Preamble(9)
strOutput = ""
'strOutput = strOutput + "Format = " + CStr(intFormat) + vbCrLf
'strOutput = strOutput + "Type = " + CStr(intType) + vbCrLf
'strOutput = strOutput + "Points = " + CStr(lngPoints) + vbCrLf
'strOutput = strOutput + "Count = " + CStr(lngCount) + vbCrLf
'strOutput = strOutput + "X increment = " + FormatNumber(dblXIncrement * 1000000) + _
'            " us" + vbCrLf
'strOutput = strOutput + "X origin = " + FormatNumber(dblXOrigin * 1000000) + _
'            " us" + vbCrLf
'strOutput = strOutput + "X reference = " + CStr(lngXReference) + vbCrLf
'strOutput = strOutput + "Y increment = " + FormatNumber(sngYIncrement * 1000) + _
'            " mV" + vbCrLf
```

```
'strOutput = strOutput + "Y origin = " + FormatNumber(sngYOrigin) + " V" + vbCrLf
'strOutput = strOutput + "Y reference = " + CStr(lngYReference) + vbCrLf
strOutput = strOutput + "Volts/Div = " + FormatNumber(lngVSteps * sngYIncrement / 8) + _
            " V" + vbCrLf
strOutput = strOutput + "Offset = " + FormatNumber(sngYOrigin) + " V" + vbCrLf
strOutput = strOutput + "Sec/Div = " + FormatNumber(lngPoints * dblXIncrement / 10 * _
            1000000) + " us" + vbCrLf
strOutput = strOutput + "Delay = " + FormatNumber(((lngPoints / 2) * _
            dblXIncrement + dblXOrigin) * 1000000) + " us" + vbCrLf

' QUERY_WAVE_DATA - Outputs waveform data that is stored in a buffer.
myScope.WriteString ":WAV:DATA?"   ' Query the oscilloscope for the waveform data.

' READ_WAVE_DATA - The wave data consists of two parts: the header,
' and the actual waveform data followed by a new line (NL) character.
' The query data has the following format:
'
'    <header><waveform_data><NL>
'
' Where:
'    <header> = #800001000 (This is an example header)
' The "#8" may be stripped off of the header and the remaining
' numbers are the size, in bytes, of the waveform data block.  The
' size can vary depending on the number of points acquired for the
' waveform.  You can then read that number of bytes from the
' oscilloscope and the terminating NL character.
'
Dim lngI As Long
Dim lngDataValue As Long

varQueryResult = myScope.ReadIEEEBlock(BinaryType_UI1)   ' Unsigned integer bytes.
For lngI = 0 To UBound(varQueryResult) Step (UBound(varQueryResult) / 20)   ' 20 points.
  If intBytesPerData = 2 Then
    lngDataValue = varQueryResult(lngI) * 256 + varQueryResult(lngI + 1)   ' 16-bit value.
  Else
    lngDataValue = varQueryResult(lngI)   ' 8-bit value.
  End If
  strOutput = strOutput + "Data point " + CStr(lngI / intBytesPerData) + ", " + _
    FormatNumber((lngDataValue - lngYReference) * sngYIncrement + sngYOrigin) + _
    " V, " + _
    FormatNumber(((lngI / intBytesPerData - lngXReference) * dblXIncrement + _
    dblXOrigin) * 1000000) + " us" + vbCrLf
Next lngI
MsgBox "Waveform data:" + vbCrLf + strOutput

' Make a delay measurement between channel 1 and 2.
Dim dblChan1Edge1 As Double
Dim dblChan2Edge1 As Double
Dim dblChan1Edge2 As Double
Dim dblDelay As Double
Dim dblPeriod As Double
Dim dblPhase As Double

myScope.WriteString ":MEASURE:TEDGE? +1, CHAN1"   ' Query time at 1st rising edge on ch1.
dblChan1Edge1 = myScope.ReadNumber   ' Read time at edge 1 on ch 1.
myScope.WriteString ":MEASURE:TEDGE? +1, CHAN2"   ' Query time at 1st rising edge on ch2.
dblChan2Edge1 = myScope.ReadNumber   ' Read time at edge 1 on ch 2.
dblDelay = dblChan2Edge1 - dblChan1Edge1   ' Calculate delay time between ch1 and ch2.
MsgBox "Delay = " + vbCrLf + CStr(dblDelay)   ' Write calculated delay time to screen.

' Make a phase difference measurement between channel 1 and 2.
myScope.WriteString ":MEASURE:TEDGE? +2, CHAN1"   ' Query time at 1st rising edge on ch1.
dblChan1Edge2 = myScope.ReadNumber   ' Read time at edge 2 on ch 1.
dblPeriod = dblChan1Edge2 - dblChan1Edge1   ' Calculate period of ch 1.
dblPhase = (dblDelay / dblPeriod) * 360 ' Calculate phase difference between ch1 and ch2.
MsgBox "Phase = " + vbCrLf + CStr(dblPhase)
```

```
  Exit Sub

VisaComError:
  MsgBox "VISA COM Error:" + vbCrLf + Err.Description

End Sub

Private Sub CheckForInstrumentErrors()

  On Error GoTo VisaComError

  Dim strErrVal As String
  Dim strOut As String

  myScope.WriteString "SYSTEM:ERROR?"   ' Query any errors data.
  strErrVal = myScope.ReadString        ' Read: Errnum,"Error String".
  While Val(strErrVal) <> 0             ' End if find: 0,"No Error".
    strOut = strOut + "INST Error: " + strErrVal
    myScope.WriteString ":SYSTEM:ERROR?"            ' Request error message.
    strErrVal = myScope.ReadString                  ' Read error message.
  Wend

  If Not strOut = "" Then
    MsgBox strOut, vbExclamation, "INST Error Messages"
    myScope.FlushWrite (False)
    myScope.FlushRead

  End If

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error: " + vbCrLf + Err.Description

End Sub
```