InfiniiVision HD3-Series Mixed Signal Oscilloscopes

KEYSIGHT

PROGRAMMER'S GUIDE

Notices

Revision

Version 10.01

Edition

September 2024

Available in electronic format only

Published by:
Keysight Technologies, Inc.
1900 Garden of the Gods Road
Colorado Springs, CO 80907 USA

Warranty

The material contained in this document is provided "as is," and is subject to being changed, without notice, in future editions. Further, to the maximum extent permitted by applicable law, Keysight disclaims all warranties, either express or implied, with regard to this manual and any information contained herein, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. Keysight shall not be liable for errors or for incidental or consequential damages in connection with the furnishing, use, or performance of this document or of any information contained herein. Should Keysight and the user have a separate written agreement with warranty terms covering the material in this document that conflict with these terms, the warranty terms in the separate agreement shall control.

Technology License

The hardware and/or software described in this document are furnished under a license and may be used or copied only in accordance with the terms of such license.

U.S. Government Rights

The Software is "commercial computer software," as defined by Federal Acquisition Regulation ("FAR") 2.101. Pursuant to FAR 12.212 and 27.405-3 and Department of Defense FAR Supplement ("DFARS") 227.7202, the U.S. government acquires commercial computer software under the same terms by which the software is customarily provided to the public. Accordingly, Keysight provides the Software to U.S. government customers under its standard commercial license, which is embodied in its End User License Agreement (EULA), a copy of which can be found at www.keysight.com/find/sweula. The license set forth in the EULA represents the exclusive authority by which the U.S. government may use, modify, distribute, or disclose the Software. The EULA and the license set forth therein, does not require or permit, among other things, that Keysight: (1) Furnish technical information related to commercial computer software or commercial computer software documentation that is not customarily provided to the public; or (2) Relinquish to, or otherwise provide, the government rights in excess of these rights customarily provided to the public to use, modify, reproduce, release, perform, display, or disclose commercial computer software or commercial computer software documentation. No additional government requirements beyond those set forth in the EULA shall apply, except to the extent that those terms, rights, or licenses are explicitly required from all providers of commercial computer software pursuant to the FAR and the DFARS and are set forth specifically in writing elsewhere in the EULA. Keysight shall be under no obligation to update, revise or otherwise modify the Software. With respect to any technical data as defined by FAR 2.101, pursuant to FAR 12.211 and 27.404.2 and DFARS 227.7102, the U.S. government acquires no greater than Limited Rights as defined in FAR 27.401 or DFAR 227.7103-5 (c), as applicable in any technical data.

Safety Notices

CAUTION

A **CAUTION** notice denotes a hazard. It calls attention to an operating procedure, practice, or the like that, if not correctly performed or adhered to, could result in damage to the product or loss of important data. Do not proceed beyond a **CAUTION** notice until the indicated conditions are fully understood and met.

WARNING

A WARNING notice denotes a hazard. It calls attention to an operating procedure, practice, or the like that, if not correctly performed or adhered to, could result in personal injury or death. Do not proceed beyond a WARNING notice until the indicated conditions are fully understood and met.

In This Book

This book is your guide to programming the HD3-Series mixed-signal oscilloscopes:

Table 1 HD3-Series Model Numbers, Bandwidths, Sample Rates

	HD302MSO	HD304MSO
Bandwidth	200 MHz standard, options for 350 MHz, 500 MHz, and 1 GHz	
Sample Rate	3.2 GSa/s	3.2 GSa/s
Memory Depth	20 Mpts standard, options for 50 Mpts and 100 Mpts	
Analog Channels	2	4
Logic Channels MSO	16	16

The first few chapters describe how to set up and get started:

- Chapter 1, “What's New,” starting on page 33, describes programming command changes in the latest version of oscilloscope software.
- Chapter 2, “Setting Up,” starting on page 49, describes the steps you must take before you can program the oscilloscope.
- Chapter 3, “Getting Started,” starting on page 57, gives a general overview of oscilloscope program structure and shows how to program the oscilloscope using a few simple examples.
- Chapter 4, “Sequential (Blocking) vs. Overlapped Commands,” starting on page 71, describes these command types and how they affect the oscilloscope and synchronization.
- Chapter 5, “Commands Quick Reference,” starting on page 73, is a brief listing of the HD3-Series oscilloscope commands and syntax.

The next chapters provide reference information on common commands, root level commands, other subsystem commands, and error messages:

- Chapter 6, “Common (*) Commands,” starting on page 179, describes commands defined by the IEEE 488.2 standard that are common to all instruments.
- Chapter 7, “Root (:) Commands,” starting on page 207, describes commands that reside at the root level of the command tree and control many of the basic functions of the oscilloscope.
- Chapter 8, “:ACQuire Commands,” starting on page 229, describes commands for setting the parameters used when acquiring and storing data.
- Chapter 9, “:BUS<n> Commands,” starting on page 249, describes commands that control all oscilloscope functions associated with the digital channels bus display.

- Chapter 10, ":CALibrate Commands," starting on page 259, describes utility commands for determining the state of the calibration factor protection button.
- Chapter 11, ":CHANnel<n> Commands," starting on page 269, describes commands that control all oscilloscope functions associated with individual analog channels or groups of channels.
- Chapter 12, ":COUNter<c> Commands," starting on page 299, describes commands that control the counter analysis feature.
- Chapter 13, ":DIGital<d> Commands," starting on page 309, describes commands that control all oscilloscope functions associated with individual digital channels.
- Chapter 14, ":DISPlay Commands," starting on page 319, describes commands that control how waveforms, graticule, and text are displayed and written on the screen.
- Chapter 15, ":DVM Commands," starting on page 363, describes commands that control the digital voltmeter analysis feature.
- Chapter 16, ":EXTernal Trigger Commands," starting on page 369, describes commands that control the input characteristics of the external trigger input.
- Chapter 17, ":FFT Commands," starting on page 375, describes commands that control the FFT function in the oscilloscope.
- Chapter 18, ":FRANalysis Commands," starting on page 401, describes commands that control oscilloscope functions associated with the Frequency Response Analysis (FRA) feature, which is available is with the license-enabled built-in waveform generator).
- Chapter 19, ":FUNCtion<m> Commands," starting on page 417, describes commands that control math waveforms.
- Chapter 20, ":HCOPy Commands," starting on page 469, describes commands that set and query the selection of hardcopy device and formatting options.
- Chapter 21, ":HISTogram Commands," starting on page 473, describes commands that control the histogram analysis feature.
- Chapter 22, ":LISTer Commands," starting on page 489, describes commands that turn on/off the Lister display for decoded serial data and get the Lister data.
- Chapter 23, ":LTESt Commands," starting on page 493, describes commands that control the Measurement Limit Test feature.
- Chapter 24, ":MARKer Commands," starting on page 507, describes commands that set and query the settings of X-axis markers (X1 and X2 cursors) and the Y-axis markers (Y1 and Y2 cursors).
- Chapter 25, ":MEASure Commands," starting on page 529, describes commands that select automatic measurements (and control markers).
- Chapter 26, ":MTESt Commands," starting on page 631, describes commands that control the license-enabled mask test feature.

- Chapter 27, ":POD Commands," starting on page 667, describes commands that control all oscilloscope functions associated with groups of digital channels.
- Chapter 28, ":RECall Commands," starting on page 675, describes commands that recall previously saved oscilloscope setups, reference waveforms, or masks.
- Chapter 29, ":SAVE Commands," starting on page 687, describes commands that save oscilloscope setups, screen images, and data.
- Chapter 30, ":SBUS<n> Commands," starting on page 715, describes commands that control oscilloscope functions associated with the serial decode bus and serial triggering.
- Chapter 31, ":SEARch Commands," starting on page 821, describes commands that control oscilloscope functions associated with searching for waveform events.
- Chapter 32, ":STATus Commands," starting on page 883, describes commands that interact with status registers.
- Chapter 33, ":SYSTem Commands," starting on page 1021, describes commands that control basic system functions of the oscilloscope.
- Chapter 34, ":TIMebase Commands," starting on page 1065, describes commands that control all horizontal sweep functions.
- Chapter 35, ":TRIGger Commands," starting on page 1077, describes commands that control the trigger modes and parameters for each trigger type.
- Chapter 36, ":WAVeform Commands," starting on page 1143, describes commands that provide access to waveform data.
- Chapter 37, ":WGEN<w> Commands," starting on page 1181, describes commands that control waveform generator (WAVEGEN license) functions and parameters.
- Chapter 38, ":WMEMory<r> Commands," starting on page 1223, describes commands that control reference waveforms.
- Chapter 39, "Obsolete and Discontinued Commands," starting on page 1233, describes obsolete commands which still work but have been replaced by newer commands and discontinued commands which are no longer supported.
- Chapter 40, "Error Messages," starting on page 1245, lists the instrument error messages that can occur while programming the oscilloscope.

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.

Then, there are chapters that describe programming topics and conceptual information in more detail:

- Chapter 41, "Status Reporting," starting on page 1253, describes the oscilloscope's status registers and how to check the status of the instrument.

- Chapter 42, “Synchronizing Acquisitions,” starting on page 1269, describes how to wait for acquisitions to complete before querying measurement results or performing other operations with the captured data.
- Chapter 43, “More About Oscilloscope Commands,” starting on page 1291, contains additional information about oscilloscope programming commands.

Finally, there is a chapter that contains programming examples:

- Chapter 44, “Programming Examples,” starting on page 1301.

See Also

- For more information on using the SICL, VISA, and VISA COM libraries in general, see the documentation that comes with the Keysight IO Libraries Suite.
- For information on oscilloscope front-panel operation, see the *User's Guide*.
- For detailed connectivity information, refer to the *Keysight Technologies USB/LAN/GPIB Connectivity Guide*. For a printable electronic copy of the *Connectivity Guide*, direct your Web browser to www.keysight.com and search for "Connectivity Guide".
- For the latest versions of this and other manuals, see: http://www.keysight.com/find/HD3-Series-manual

Contents

13 :DIGital<d> Commands

14 :DISPlay Commands

26 :MTESt Commands

31 :SEARch Commands

33 :SYSTem Commands

37 :WGEN<w> Commands

1 What's New

Version 10.00 at Introduction

The Keysight InfiniiVision HD3-Series mixed signal oscilloscopes were introduced with version 10.00 of oscilloscope operating software.

The command set is most closely related to the InfiniiVision 3000G X-Series oscilloscopes (and the 3000/3000T X-Series, 7000A/B Series, 6000 Series, and 54620/54640 Series oscilloscopes before them). For more information, see "Command Differences From 3000G X-Series Oscilloscopes" on page 35.

Command Differences From 3000G X-Series Oscilloscopes

The Keysight InfiniiVision HD3-Series mixed signal oscilloscopes command set is most closely related to the InfiniiVision 3000G X-Series oscilloscopes (and the 3000/3000T X-Series, 7000A/B Series, 6000 Series, and 54620/54640 Series oscilloscopes before them).

The main differences between the version 10.00 programming command set for the InfiniiVision HD3-Series oscilloscopes and the 7.60 programming command set for the InfiniiVision 3000G X-Series oscilloscopes are related to:

- There is no Edge then Edge trigger mode in the HD3-Series oscilloscopes.
- There is no Nth Edge Burst trigger mode in the HD3-Series oscilloscopes.
- There is no TV trigger mode in the HD3-Series oscilloscopes.
- There is no USB 2.0 signal quality analysis feature or USB trigger in the HD3-Series oscilloscopes.
- There is no power measurements application feature in the HD3-Series oscilloscopes.
- Up to 20 annotations can be defined in the InfiniiVision HD3-Series oscilloscopes as compared to 10 annotations in the InfiniiVision 3000G X-Series oscilloscopes, and annotations now have a mode and can be assigned to a grid or a waveform source.
- There can be up to four graticules on the display in the InfiniiVision HD3-Series oscilloscopes as compared to one graticule in the InfiniiVision 3000G X-Series oscilloscopes.
- There are now commands for displaying the measurement results catalog, layout, and size.
- Status registers and the commands for interacting with them are now SCPI 99 compliant.
- There are two counters instead of one.
- :HARDcopy commands for setting up network printers are not included. Everything you could print in the 3000G X-Series oscilloscopes can be saved to a file on a USB storage device.
- CAN XL is supported by the CAN protocol decode.
- The ARINC 429, CXPI, FlexRay, I2S, MIL-STD-1553, Manchester, NRZ, SENT, USB, and USB PD protocol decodes are not available in the HD3-Series oscilloscopes.
- The HD3-Series oscilloscopes have a new SCPI parser with some behavior differences as compared to the SCPI parser in the 3000G X-Series oscilloscopes: see "Commands Not Supported in Multiple Program Message Units" on page 1297 and "Queries Not Supported in Multiple Program Message Units" on page 1297.

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

New Commands

Command	Description
:ACQuire:BANDwidth (see page 232)	Sets the global bandwidth limit or turns it off. Lower bandwidth limits turn on the high-definition improved vertical resolution.
:DIGital:ORDer:ASCending (see page 311)	Displays the digital channels that are turned on in ascending order.
:DIGital:ORDer:DESCending (see page 312)	Displays the digital channels that are turned on in descending order.
:DIGital:ORDer[:SET] (see page 313)	Displays digital channels in the order listed or returns the current order of displayed digital channels.
:DIGital<d>:SETup? (see page 316)	Returns setup information for the specified digital channel.
:DISPlay:ANNotation<n>:GRID (see page 327)	When an annotation's mode is GRID (see :DISPlay:ANNotation<n>:MODE), this command makes the association to a grid.
:DISPlay:ANNotation<n>:MODE (see page 328)	Specifies whether an annotation is associated with a grid or a waveform source.
:DISPlay:ANNotation<n>:SOURce (see page 329)	When an annotation's mode is SOURce (see :DISPlay:ANNotation<n>:MODE), this command makes the association to a waveform source.
:DISPlay:ANNotation<n>:XPOSition (see page 331)	Sets the annotation's horizontal X position.
:DISPlay:ANNotation<n>:YPOSition (see page 332)	Sets the annotation's vertical Y position.
:DISPlay:CLOCk:IGRid (see page 335)	Specifies whether the oscilloscope's clock information is displayed in the first waveform grid (ON) or in the badges bar at the top of the screen (OFF).
:DISPlay:CLOCk:IGRid:XPOSition (see page 336)	Specifies the horizontal X position of the clock information as a percent of the grid width.
:DISPlay:CLOCk:IGRid:YPOSition (see page 337)	Specifies the vertical Y position of the clock information as a percent of the grid height.
:DISPlay:CLOCk[:STATe] (see page 338)	Enables or disables the display of the oscilloscope's clock information.
:DISPlay:GRATicule:COUNt (see page 343)	When the graticule (grid) layout is set to CUSTom (see :DISPlay:GRATicule:LAYout), this command specifies the number of waveform grids.
:DISPlay:GRATicule:ALABels:DUAL (see page 341)	Specifies whether grid scales are shown for two waveforms.

Command	Description
:DISPlay:GRATicule:ALABels:IGRid (see page 342)	Specifies whether graticule (grid) axis labels are displayed in separate scale bars (OFF) or within the grid (ON).
:DISPlay:GRATicule:LAYout (see page 343)	Specifies the layout of waveform display grids.
:DISPlay:GRATicule:SET (see page 343)	When the graticule (grid) layout is set to CUSTom (see :DISPlay:GRATicule:LAYout), this command assigns a waveform source to a particular grid.
:DISPlay:GRATicule:SOURce? (see page 343)	Returns a comma-separated list of the waveforms in the grid.
:DISPlay:RESults:CATalog? (see page 355)	Returns a comma-separated list of the windows in the results area.
:DISPlay:RESults:LAYout (see page 356)	Specifies the layout of windows in the results area (TAB or LIST).
:DISPlay:RESults:LAYout:LIST[:SELect] (see page 357)	Lets you select and highlight a particular window in the results area when windows are listed.
:DISPlay:RESults:LAYout:TAB:LEFT[:SELect] (see page 358)	Lets you select and highlight a particular window in the results area left pane when windows are tabbed.
:DISPlay:RESults:LAYout:TAB:RIGHt[:SELect] (see page 359)	Lets you select and highlight a particular window in the results area right pane when windows are tabbed.
:DISPlay:RESults:SIZE (see page 360)	Specifies the size of the results area.
:FFT:BSIZe? (see page 379)	Returns the FFT's bin size value.
:FFT:DETection:POINts (see page 382)	Specifies the maximum number of points that FFT detectors should decimate to.
:FFT:DETection:TYPE (see page 383)	Lets you set the FFT detector decimation type for FFT (Magnitude) functions or specify that no detector is used.
:FFT:DMODe (see page 385)	Selects the FFT waveform display mode.
:FFT:RBWidth? (see page 391)	Returns the FFT's resolution bandwidth value.
:FFT:READout (see page 392)	Specifies whether the FFT resolution is displayed in the FFT spectrum plot, and if it is, whether the readout shows sample rate, bin size, or resolution bandwidth.
:FFT:SRATe? (see page 397)	Returns the FFT's sample rate value.
:FUNCtion<m>:LABel (see page 452)	Sets or queries a math function's label string.
:MEASure:QUICk (see page 589)	Installs 10 common measurements on an analog input channel source.

Command	Description
:MEASure:THResholds:ABSolute (see page 612)	Sets the upper, middle, and lower measurement thresholds to absolute values.
:MEASure:THResholds:METHod (see page 613)	Specifies whether measurement threshold values are entered as percent levels between Vbase and Vtop or as absolute (static) values.
:MEASure:THResholds:PERCent (see page 614)	Sets the upper, middle, and lower measurement thresholds to user-defined percentages between 5% and 95% of values between Vbase and Vtop.
:MEASure:THResholds:STANdard (see page 615)	Sets the lower, middle, and upper measurement thresholds to 10%, 50%, and 90% values between Vbase and Vtop.
:POD<n>:NIBBle<n>:THReshold (see page 670)	Sets the logic threshold value for a nibble within a pod.
:SBUS<n>:CAN:SIGNal:XLBaudrate (see page 735)	Sets the CAN XL baud rate.
:SBUS<n>:CAN:TRIGger:TYPE (see page 750)	Selects the CAN trigger type.
:SBUS<n>:CAN:TYPE (see page 751)	Selects the CAN protocol decode type.
:SBUS<n>:CAN:XLSPoint (see page 752)	Sets the point during the CAN XL bit time where the bit level is sampled to determine whether the bit is dominant or recessive.
:STATus Commands (see page 883)	Subsystem for controlling status registers.
:SYSTem:ERRor:ALL? (see page 1028)	Returns a comma-separated list of all errors from the queue (codes and strings).
:SYSTem:ERRor:CODE[:NEXT]? (see page 1029)	Returns the next error (code only) from the queue.
:SYSTem:ERRor:COUNt? (see page 1030)	Returns an integer number of errors in the queue.
:SYSTem:ERRor:EXTended? (see page 1031)	Returns a comma-separated list of all errors (strings, no codes) of the specified type from the queue.
:SYSTem:HELP:HEADers? (see page 1034)	Returns a binary block that contains a newline-separated list of supported SCPI commands and queries.
:SYSTem:HID? (see page 1034)	Returns the oscilloscope's Host ID string (the same as :SYSTem:DIDentifier?).
:SYSTem:POWercycle (see page 1042)	Powers-down the oscilloscope and restarts it. This is the same as pressing the front panel power button to power-down the oscilloscope and then pressing the power button again to start the oscilloscope.
:SYSTem:SHUTdown (see page 1057)	Powers-down the oscilloscope. This is the same as pressing the front panel power button to power-down the oscilloscope.

Command	Description
:SYSTem:TIME:DSAVing (see page 1059)	Enables or disables the "automatically adjust clock for Daylight Savings Time" setting.
:SYSTem:TIME:NTPRotocol (see page 1060)	Enables or disables automatically setting the oscilloscope's clock using Network Time Protocol.
:SYSTem:TIME:ZONE (see page 1061)	Sets the system time zone based on the specified offset from UTC.
:WGEN<w>:DCMode (see page 1195)	When the DC waveform type is selected, this command lets you choose between PRECise output levels using a smaller range (±1.00 V) or a WIDerange of voltages (±8.00 V).

Changed Commands

Command	Differences From InfiniiVision 3000G X-Series Oscilloscopes
:ACQuire:TYPE (see page 247)	There is no HRESolution acquisition type. Instead, higher vertical resolutions are achieved by selecting lower global bandwidth limits in the :ACQuire:BANDwidth (see page 232) command.
:CALibrate:LABel (see page 262)	The :CALibrate:LABel? query now returns the string within quotes.
:CALibrate:PROTected (see page 264)	There is now a command form that disables or enables the Cal Protect control which allows user calibration to be run or prevents user calibration from being run.
:DISPlay:DATA? (see page 339)	The option for selecting a color or grayscale palette is no longer available. Also, the BMP8bit option is no longer available.
:MTESt:TITLe? (see page 665)	The query now returns the string within quotes.
*RCL (see page 193)	The *RCL command is similar to the :RECall:SETup[:STARt] (see page 684) command and will recall setup files from internal locations under the "/User Files/" folder as well as from external USB storage device locations under the "/USB1/" (or "/usb/"), "/USB2/", or "/USB3/" folders depending on the number of storage devices connected.
:RECall:ARBitrary[:STARt] (see page 678)	When recalling from internal locations, quoted ASCII file name strings beginning with "/User Files/" are used instead of internal location numbers.
:RECall:MASK[:STARt] (see page 682)	When recalling from internal locations, quoted ASCII file name strings beginning with "/User Files/" are used instead of internal location numbers.
:RECall:SETup[:STARt] (see page 684)	When recalling from internal locations, quoted ASCII file name strings beginning with "/User Files/" are used instead of internal location numbers.

Command	Differences From InfiniiVision 3000G X-Series Oscilloscopes
*SAV (see page 197)	The *SAV command is similar to the :SAVE:SETup[:STARt] (see page 707) command and will save setup files to internal locations under the "/User Files/" folder as well as to external USB storage device locations under the "/USB1/" (or "/usb/"), "/USB2/", or "/USB3/" folders depending on the number of storage devices connected.
:SAVE:ARBitrary[:STARt] (see page 691)	When saving to internal locations, quoted ASCII file name strings beginning with "/User Files/" are used instead of internal location numbers.
:SAVE:MASK[:STARt] (see page 697)	When saving to internal locations, quoted ASCII file name strings beginning with "/User Files/" are used instead of internal location numbers.
:SAVE:SETup[:STARt] (see page 707)	When saving to internal locations, quoted ASCII file name strings beginning with "/User Files/" are used instead of internal location numbers.
:SBUS<n>:CAN:TRIGger (see page 737)	Separate FDData and FDMSignal options are no longer necessary with the addition of the :SBUS<n>:CAN:TRIGger:TYPE command.
:SBUS<n>:IIC:TRIGger[:TYPE] (see page 761)	The "READEprom" option has changed to "REPRom".
:SEARch:SERial:IIC:MODE (see page 858)	The "READEprom" option has changed to "REPRom".
:SERial? (see page 222)	The query now returns the string within quotes.
:SYSTem:ERRor[:NEXT]? (see page 1032)	The query is the same except the optional [:NEXT] can be included.
:SYSTem:PERSona[:MANufacturer] (see page 1040)	The :SYSTem:PERSona[:MANufacturer]? query now returns the string within quotes.

Discontinued Commands

Discontinued commands are commands that were available in InfiniiVision 3000G X-Series oscilloscopes, but are not supported by the InfiniiVision HD3-Series oscilloscopes. Listed below are the Discontinued commands and the nearest equivalent command available (if any).

Discontinued Command	Current Command Equivalent	Comments
:ACQuire:AALias?	none	Anti-alaising is always present in the HD3-Series oscilloscopes.
:ACQuire:DAALias	none	There is no option to turn off anti-alaising in the HD3-Series oscilloscopes.

Discontinued Command	Current Command Equivalent	Comments
:CHANnel2:SKEW	:CHANnel<n>:PROBe:SKEW (see page 291)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:CHANnel:ACTivity	:ACTivity (see page 210)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:CHANnel:LABel	:CHANnel<n>:LABel (see page 278) or :DIGital<n>:LABel (see page 315)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:CHANnel:THReshold	:POD<n>:THReshold (see page 673) or :DIGital<d>:THReshold (see page 318)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:CHANnel<n>:BANDwidth	:ACQuire:BANDwidth (see page 232) or :CHANnel<n>:BWLimit (see page 273)	The the HD3-Series oscilloscopes have a configurable global bandwidth limit (see :ACQuire:BANDwidth) and a per-channel 40 MHz bandwidth limit (see :CHANnel:BWLimit).
:CHANnel<n>:INPut	:CHANnel<n>:IMPedance (see page 276)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:CHANnel<n>:PMODe	none	This was an obsolete command in the 3000G X-Series oscilloscopes.
:DIGital<d>:POSition	:DIGital:ORDer[:SET] (see page 313)	The notion of digital channel position locations is not present in the HD3-Series oscilloscopes. You can, however, order digital channels using the :DIGital:ORDer[:SET] (see page 313) command.
:DISPlay:ANNotation<n>:X1Position	:DISPlay:ANNotation<n>:XPOSition (see page 331)	Discontinued command positioned by pixel position, current command positions by percent of grid width.
:DISPlay:ANNotation<n>:Y1Position	:DISPlay:ANNotation<n>:YPOSition (see page 332)	Discontinued command positioned by pixel position, current command positions by percent of grid height.

Discontinued Command	Current Command Equivalent	Comments
:DISPlay:CONNect	none	This was an obsolete command in the 3000G X-Series oscilloscopes.
:DISPlay:GRATicule:TYPE	none	This command was for the TV trigger mode which is not available in the HD3-Series oscilloscopes.
:DISPlay:MENU	none	On earlier InfiniiVision oscilloscopes, this command would display a particular softkey menu. The HD3-Series oscilloscopes do not have softkey menus.
:DISPlay:ORDer	none	This was an obsolete command in the 3000G X-Series oscilloscopes.
:DISPlay:SIDebar	none	On earlier InfiniiVision oscilloscopes, this command would display a particular sidebar dialog box. The HD3-Series oscilloscopes do not have sidebar dialog boxes.
:DISPlay:VECTors	none	On the 3000G X-Series oscilloscopes, lines connecting acquisition points was always on. Because it is also always on in the HD3-Series oscilloscopes, this command is not necessary.
:DVM:FREQuency?	:COUNter<c>:CURRent? (see page 301)	The :DVM:FREQuency? query has been replaced by the new Chapter 12, ":COUNter<c> Commands," starting on page 299.
:ERASe	:DISplay:CLEar (see page 334)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:EXTernal:PMODe	none	This was an obsolete command in the 3000G X-Series oscilloscopes.

Discontinued Command	Current Command Equivalent	Comments
:FFT:DMODe	none	FFT display mode options are not available in the HD3-Series oscilloscopes. However, You can enable an FFT Max Function in the user interface.
:FUNCtion:GOFT:OPERation	:FUNCtion1:OPERation (see page 456)	GOFT maps to FUNCtion1. This was an obsolete command in the 3000G X-Series oscilloscopes.
:FUNCtion:GOFT:SOURce1	:FUNCtion1:SOURce1 (see page 464)	GOFT maps to FUNCtion1. This was an obsolete command in the 3000G X-Series oscilloscopes.
:FUNCtion:GOFT:SOURce2	:FUNCtion1:SOURce2 (see page 466)	GOFT maps to FUNCtion1. This was an obsolete command in the 3000G X-Series oscilloscopes.
:FUNCtion:SOURce	:FUNCtion:SOURce1 (see page 464)	Obsolete command has ADD, SUBTract, and MULTiply parameters; current command has GOFT parameter. This was an obsolete command in the 3000G X-Series oscilloscopes.
:FUNCtion:VIEW	:FUNCtion:DISPlay (see page 431)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:HARDcopy:DESTination	:RECall:FILename (see page 680) :SAVE:FILename (see page 680)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:HARDcopy:FILename	:RECall:FILename (see page 680) :SAVE:FILename (see page 680)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:HARDcopy:GRAYscale	none	This was an obsolete command in the 3000G X-Series oscilloscopes.
:HARDcopy:IGColors	none	This was an obsolete command in the 3000G X-Series oscilloscopes.
:HARDcopy:PDRiver	none	This was an obsolete command in the 3000G X-Series oscilloscopes.

Discontinued Command	Current Command Equivalent	Comments
:HWEenable	:STATus:OPERation:HARDware:ENABle (see page 925)	Status registers now use SCPI standard commands and queries for controlling and reading the registers.
:HWERegister:CONDition?	:STATus:OPERation:HARDware:CONDition? (see page 924)	
:HWERegister[:EVENt]?	:STATus:OPERation:HARDware[:EVENt]? (see page 928)	
:MEASure:ALL	:MEASure:QUICk (see page 589)	There is no "Snapshot All" feature in the HD3-Series oscilloscopes, but there is a "Quick Measure" that will install 10 common measurements on an analog input channel source.
:MEASure:DEFine	:MEASure:DELay:DEFine (see page 555) :MEASure:THResholds:ABSolute (see page 612) :MEASure:THResholds:METHod (see page 613) :MEASure:THResholds:PERCent (see page 614) :MEASure:THResholds:STANdard (see page 615)	The :MEASure:DEFine functionality is provided by the current command equivalents.
:MEASure:LOWer	:MEASure:THResholds:ABSolute (see page 612) :MEASure:THResholds:METHod (see page 613) :MEASure:THResholds:PERCent (see page 614) :MEASure:THResholds:STANdard (see page 615)	The :MEASure:THResholds commands can define absolute values or percentage. This was an obsolete command in the 3000G X-Series oscilloscopes.
:MEASure:TDELta	:MARKer:XDELta (see page 518)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:MEASure:THResholds	:MEASure:THResholds:ABSolute (see page 612) :MEASure:THResholds:METHod (see page 613) :MEASure:THResholds:PERCent (see page 614) :MEASure:THResholds:STANdard (see page 615)	The :MEASure:THResholds commands can define absolute values or percentage. This was an obsolete command in the 3000G X-Series oscilloscopes.

Discontinued Command	Current Command Equivalent	Comments
:MEASure:TMAX	:MEASure:XMAX (see page 628)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:MEASure:TMIN	:MEASure:XMIN (see page 629)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:MEASure:TSTArt	:MARKer:X1Position (see page 513)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:MEASure:TSTOp	:MARKer:X2Position (see page 516)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:MEASure:TVOLt	:MEASure:TVALue (see page 616)	TVALue measures additional values such as db, Vs, etc. This was an obsolete command in the 3000G X-Series oscilloscopes.
:MEASure:UPPer	:MEASure:THResholds:ABSolute (see page 612) :MEASure:THResholds:METHod (see page 613) :MEASure:THResholds:PERCent (see page 614) :MEASure:THResholds:STANdard (see page 615)	The :MEASure:THResholds commands can define absolute values or percentage. This was an obsolete command in the 3000G X-Series oscilloscopes.
:MEASure:VDELta	:MARKer:YDELta (see page 525)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:MEASure:VSTArt	:MARKer:Y1Position (see page 522)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:MEASure:VSTOp	:MARKer:Y2Position (see page 524)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:MEASure:VTIMe	:MEASure:YATX (see page 630)	With :MEASure:YATX, there is a command for installing the measurement on the oscilloscope's display. :MEASure:VTIMe had no command, only a query.

Discontinued Command	Current Command Equivalent	Comments
:MTEenable	:STATus:OPERation:MTESt:ENABle (see page 951)	Status registers now use SCPI standard commands and queries for controlling and reading status registers.
:MTERegister[:EVENt]?	:STATus:OPERation:MTESt[:EVENt]? (see page 954)	
:MTESt:AMASk:{SAVE | STORe}	:SAVE:MASK[:STARt] (see page 697)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:MTESt:AVERage	:ACQuire:TYPE AVERage (see page 247)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:MTESt:AVERage:COUNt	:ACQuire:COUNt (see page 234)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:MTESt:LOAD	:RECall:MASK[:STARt] (see page 682)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:MTESt:RUMode	:MTESt:RMODe (see page 651)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:MTESt:RUMode:SOFailure	:MTESt:RMODe:FACTion:STOP (see page 655)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:MTESt:{STARt | STOP}	:RUN (see page 221) or :STOP (see page 225)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:MTESt:TRIGger:SOURce	:TRIGger Commands (see page 1077)	There are various commands for setting the source with different types of triggers. This was an obsolete command in the 3000G X-Series oscilloscopes.
:OPEE	:STATus:OPERation:ENABle (see page 899)	Status registers now use SCPI standard commands and queries for controlling and reading status registers.
:OPERegister:CONDition?	:STATus:OPERation:CONDition? (see page 898)	
:OPERegister[:EVENt]?	:STATus:OPERation[:EVENt]? (see page 902)	
:OVLenable	:STATus:OPERation:OVERload:ENABle (see page 964)	Status registers now use SCPI standard commands and queries for controlling and reading status registers.
:OVLRegister?	:STATus:OPERation:OVERload[:EVENt]? (see page 967)	

Discontinued Command	Current Command Equivalent	Comments
:PRINt	:DISPlay:DATA? (see page 339)	Printers are not supported in the HD3-Series oscilloscopes. Everything you could print in the 3000G X-Series oscilloscopes can be saved to a file on a USB storage device. The :PRINt? query was a obsolete in the 3000G X-Series oscilloscopes.
:SAVE:IMAGe:AREA	none	This was an obsolete command in the 3000G X-Series oscilloscopes.
:SAVE:IMAGe:PALette	none	Color images are always saved in the HD3-Series oscilloscopes.
:SBUS<n>:CAN:FDSTandard	none	This is no longer an option. The ISO standard is used when decoding or triggering on FD frames.
:SBUS<n>:LIN:SIGNal:DEFinitio n	none	This was an obsolete command in the 3000G X-Series oscilloscopes.
:SYSTem:MENU	none	This was an obsolete command in the 3000G X-Series oscilloscopes.
:SYSTem:PRECision	:SYSTem:PRECision:LENGth (see page 1043)	Precision analysis is no longer a feature to be enabled. Instead, you simply choose the desired analysis record length.
:TRIGger:THReshold	:POD<n>:THReshold (see page 673) or :DIGital<d>:THReshold (see page 318)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:TRIGger:ZONE:STATe	none	There is no longer an overall zone trigger feature on/off state.
:WAVeform:VIEW	none	In the HD3-Series oscilloscopes, all captured data is used.

2 Setting Up

This chapter explains how to install the Keysight IO Libraries Suite software, connect the oscilloscope to the controller PC, set up the oscilloscope, and verify the oscilloscope connection.

Step 1. Install Keysight IO Libraries Suite software

1. Download the Keysight IO Libraries Suite software from the Keysight web site at:
   - http://www.keysight.com/find/iolib
2. Run the setup file, and follow its installation instructions.

Step 2. Connect and set up the oscilloscope

The HD3-Series oscilloscope has two different interfaces you can use for programming:

- USB (device port).
- LAN. To configure the LAN interface, see "Using the LAN Interface" on page 51.

These interfaces are always active.

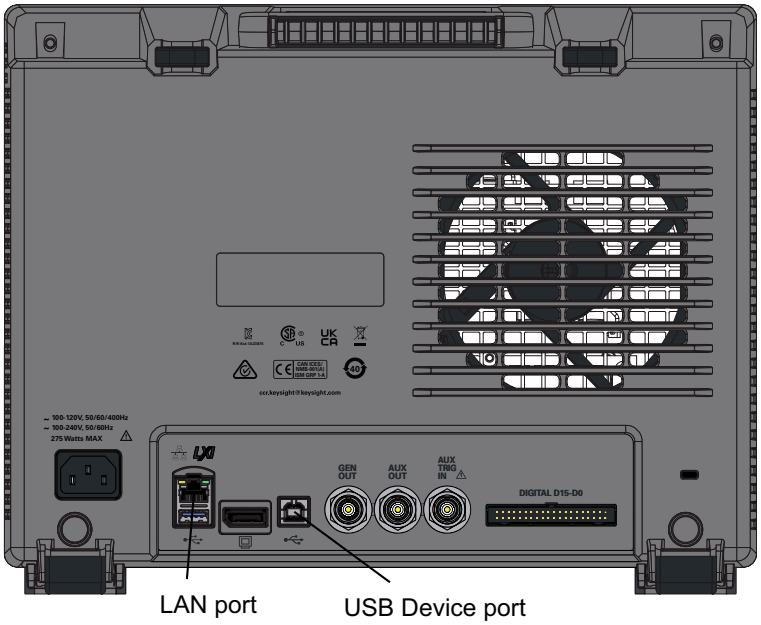


Figure 1 Control Connectors on Rear Panel

Using the USB (Device) Interface

1 Connect a USB cable from the controller PC's USB port to the "USB DEVICE" port on the back of the oscilloscope.

This is a USB 2.0 high-speed port.

Using the LAN Interface

1 If the controller PC is not already connected to the local area network (LAN), do that first.

2 Contact your network administrator about adding the oscilloscope to the network.

Find out if automatic configuration via DHCP or AutoIP can be used. Also, find out whether your network supports Dynamic DNS or Multicast DNS.

If automatic configuration is not supported, get the oscilloscope's network parameters (hostname, domain, IP address, subnet mask, gateway IP, DNS IP, etc.).

3 Connect the oscilloscope to the local area network (LAN) by inserting LAN cable into the "LAN" port.

4 Configure the oscilloscope's LAN interface:

   a From the menu, choose **Utilities > I/O...**.

   b In the I/O dialog box's **LAN Setup** tab, select **LAN Settings...**.

   c In the LAN Settings dialog box:

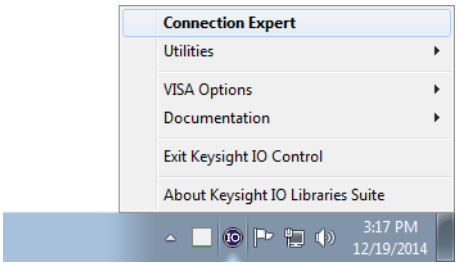
      - If automatic configuration via DHCP or AutoIP can be used, select **Automatic**.
      - If your network supports Multicast DNS, select **Multicast DNS**.
      - If automatic configuration is not supported, clear **Automatic**. Then, enter the appropriate values in the **IP Address**, **Subnet Mask**, **Gateway IP**, and **DNS IP** fields.

   d Enter the oscilloscope's **Host Name**.

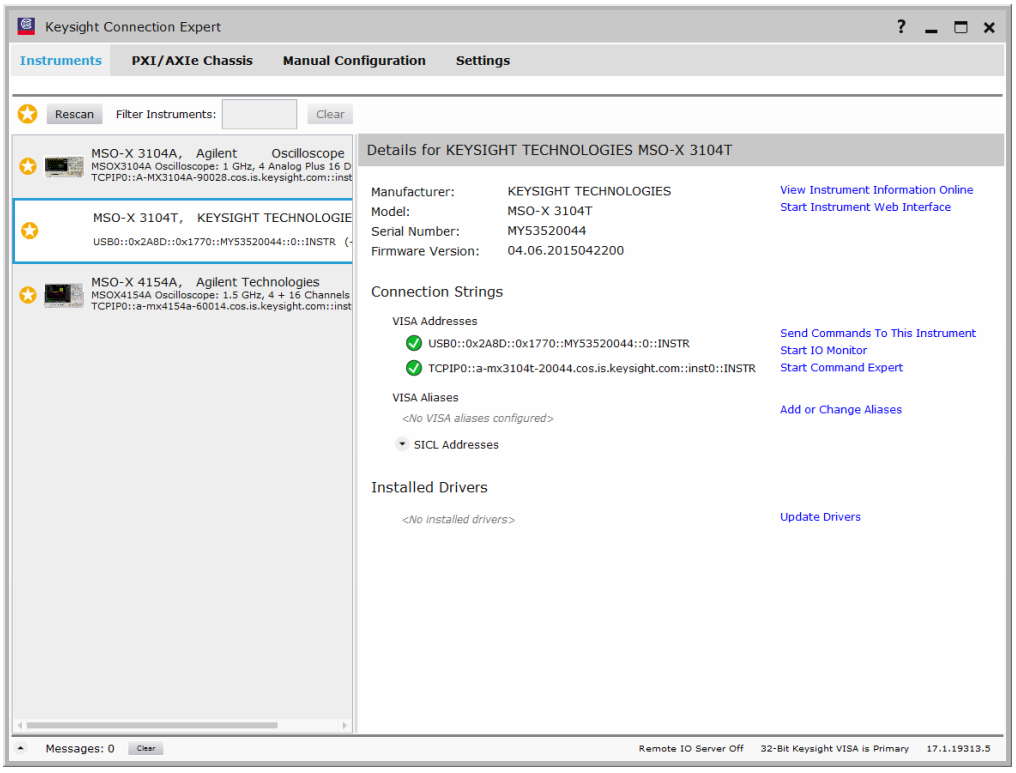
   e Select **Apply**.

Step 3. Verify the oscilloscope connection

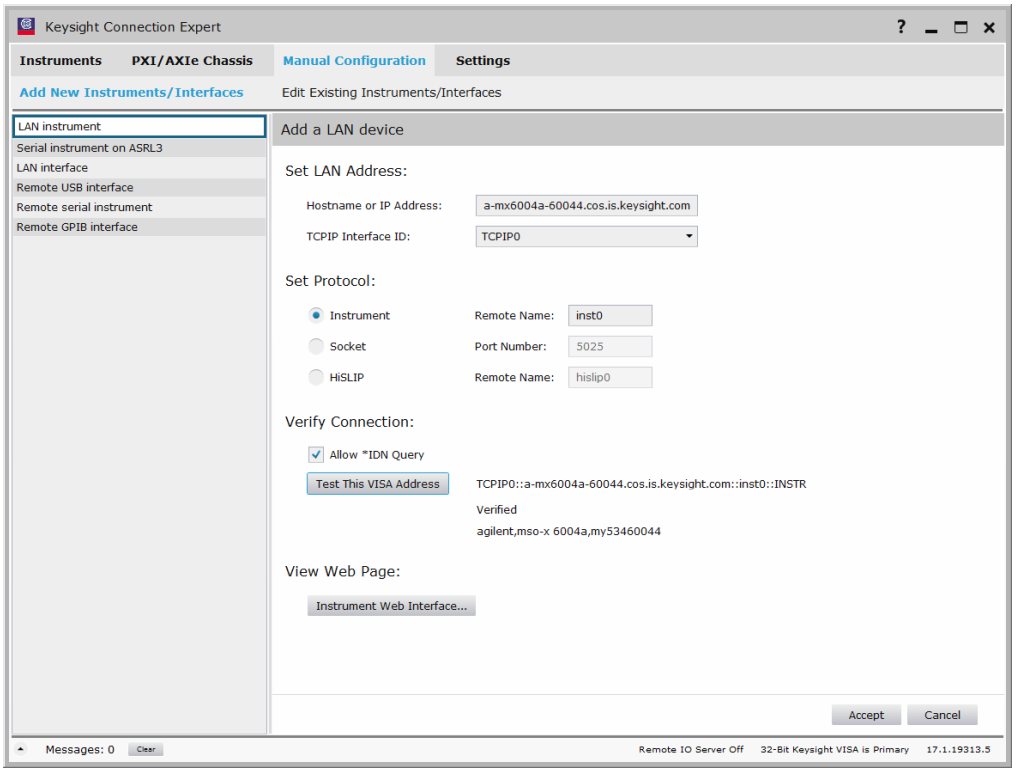
1 On the controller PC, click on the Keysight IO Control icon in the taskbar and choose **Connection Expert** from the popup menu.



2 In the Keysight Connection Expert application, instruments connected to the controller's USB interface as well as instruments on the same LAN subnet should automatically appear in the Instruments tab.



3 If your instrument does not appear, you can add it using the Manual Configuration tab.



For example, to add a device:

a Select **LAN instrument** in the list on the left.

b Enter the oscilloscope's **Hostname** or **IP address**.

c Select the protocol.

d Select **Instrument** under Set Protocol.

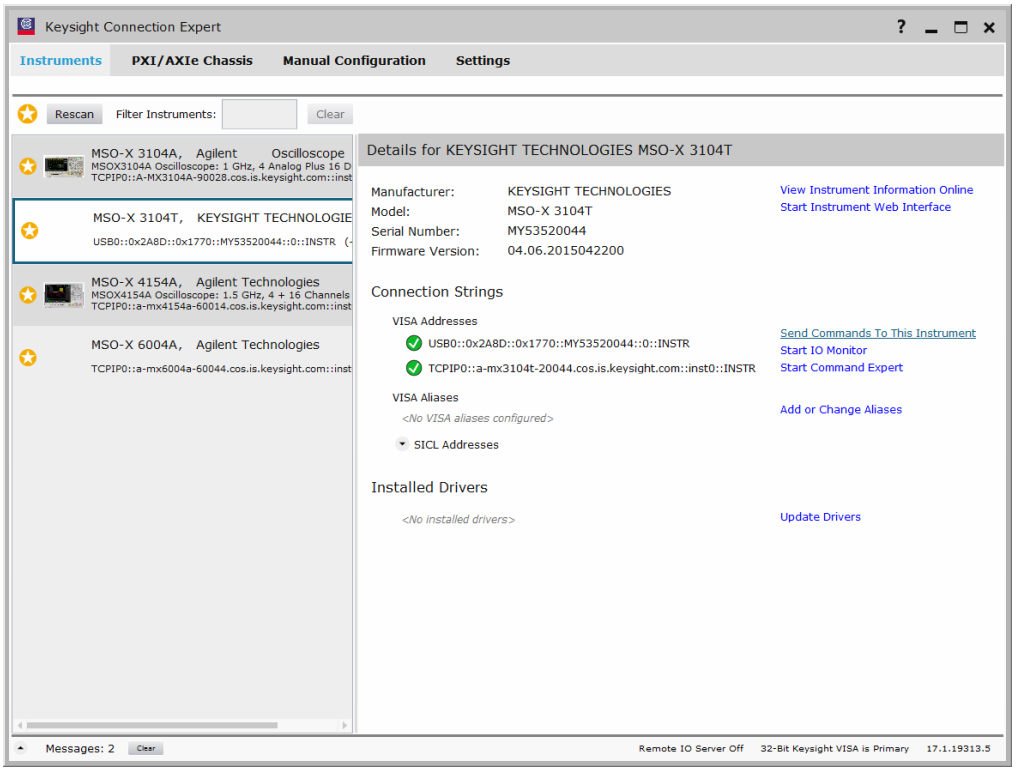
e Click **Test This VISA Address** to verify the connection.

f If the connection test is successful, click **Accept** to add the instrument.

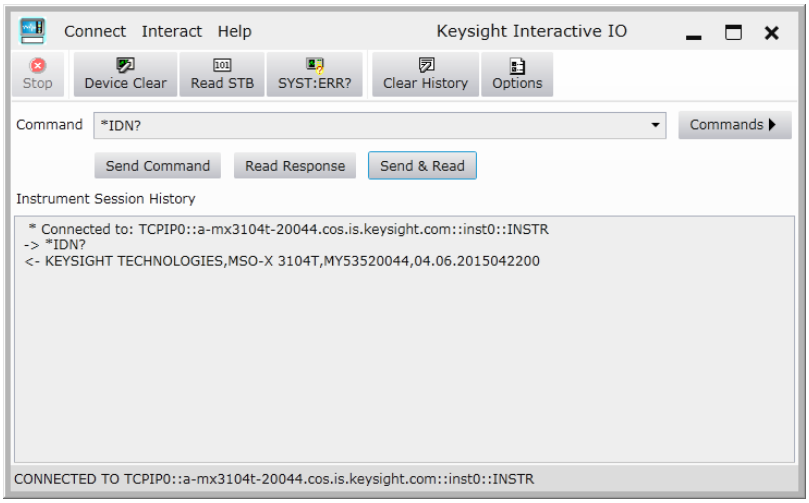
If the connection test is not successful, go back and verify the LAN connections and the oscilloscope setup.

4 Test some commands on the instrument:

a In the Details for the selected instrument, click **Send Commands To This Instrument**.



b In the Keysight Interactive IO application, enter commands in the **Command** field and press **Send Command**, **Read Response**, or **Send & Read**.



c Choose **Connect > Exit** from the menu to exit the Keysight Interactive IO application.

5 In the Keysight Connection Expert application, choose **File > Exit** from the menu to exit the application.

3 Getting Started

This chapter gives you an overview of programming the HD3-Series oscilloscopes. It describes basic oscilloscope program structure and shows how to program the oscilloscope using a few simple examples.

The getting started examples show how to send oscilloscope setup, data capture, and query commands, and they show how to read query results.

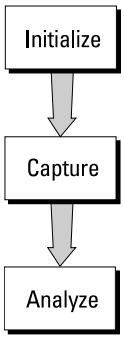
NOTE

Language for Program Examples

The programming examples in this guide are written in Visual Basic using the Keysight VISA COM library.

Basic Oscilloscope Program Structure

The following figure shows the basic structure of every program you will write for the oscilloscope.



Initializing

To ensure consistent, repeatable performance, you need to start the program, controller, and oscilloscope in a known state. Without correct initialization, your program may run correctly in one instance and not in another. This might be due to changes made in configuration by previous program runs or from the front panel of the oscilloscope.

- Program initialization defines and initializes variables, allocates memory, or tests system configuration.
- Controller initialization ensures that the interface to the oscilloscope is properly set up and ready for data transfer.
- Oscilloscope initialization sets the channel configuration, channel labels, threshold voltages, trigger specification, trigger mode, timebase, and acquisition type.

Capturing Data

Once you initialize the oscilloscope, you can begin capturing data for analysis. Remember that while the oscilloscope is responding to commands from the controller, it is not performing acquisitions. Also, when you change the oscilloscope configuration, any data already captured will most likely be rendered.

To collect data, you use the :DIGitize command. This command clears the waveform buffers and starts the acquisition process. Acquisition continues until acquisition memory is full, then stops. The acquired data is displayed by the oscilloscope, and the captured data can be measured, stored in acquisition memory in the oscilloscope, or transferred to the controller for further analysis. Any additional commands sent while :DIGitize is working are buffered until :DIGitize is complete.

You could also put the oscilloscope into run mode, then use a wait loop in your program to ensure that the oscilloscope has completed at least one acquisition before you make a measurement. Keysight does not recommend this because the needed length of the wait loop may vary, causing your program to fail. :DIGitize, on the other hand, ensures that data capture is complete. Also, :DIGitize, when complete, stops the acquisition process so that all measurements are on displayed data, not on a constantly changing data set.

Analyzing Captured Data

After the oscilloscope has completed an acquisition, you can find out more about the data, either by using the oscilloscope measurements or by transferring the data to the controller for manipulation by your program. Built-in measurements include: frequency, duty cycle, period, positive pulse width, and negative pulse width.

Using the :WAVeform commands, you can transfer the data to your controller. You may want to display the data, compare it to a known good measurement, or simply check logic patterns at various time intervals in the acquisition.

Programming the Oscilloscope

Referencing the IO Library

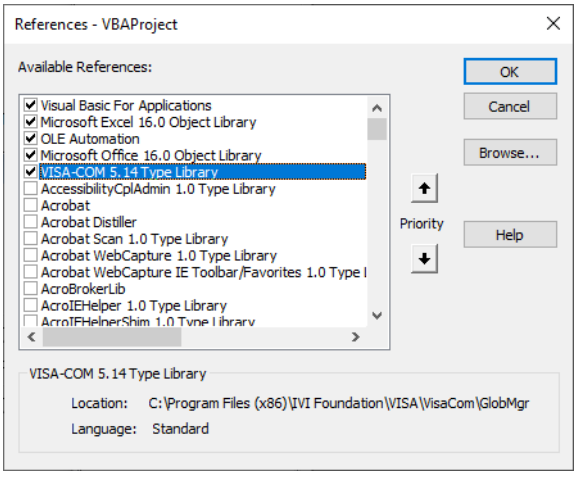
No matter which instrument programming library you use (SICL, VISA, or VISA COM), you must reference the library from your program.

In C/C++, you must tell the compiler where to find the include and library files (see the Keysight IO Libraries Suite documentation for more information).

To reference the Keysight VISA COM library in Visual Basic for Applications (VBA, which comes with Microsoft Office products like Excel):

1 Choose **Tools > References...** from the main menu.

2 In the References dialog, check the "VISA COM 5.14 Type Library".



3 Click **OK**.

To reference the Keysight VISA COM library in Microsoft Visual Basic 6.0:

1 Choose **Project > References...** from the main menu.

2 In the References dialog, check the "VISA COM 5.14 Type Library".

3 Click **OK**.

Opening the Oscilloscope Connection via the IO Library

PC controllers communicate with the oscilloscope by sending and receiving messages over a remote interface. Once you have opened a connection to the oscilloscope over the remote interface, programming instructions normally appear as ASCII character strings embedded inside write statements of the programing language. Read statements are used to read query responses from the oscilloscope.

For example, when using the Keysight VISA COM library in Visual Basic (after opening the connection to the instrument using the ResourceManager object's Open method), the FormattedIO488 object's WriteString, WriteNumber, WriteList, or WriteIEEEBlock methods are used for sending commands and queries. After a query is sent, the response is read using the ReadString, ReadNumber, ReadList, or ReadIEEEBlock methods.

The following Visual Basic statements open the connection and send a command that turns on the oscilloscope's label display.

```
Dim myMgr As VisaComLib.ResourceManager
Dim myScope As VisaComLib.FormattedIO488

Set myMgr = New VisaComLib.ResourceManager
Set myScope = New VisaComLib.FormattedIO488

' Open the connection to the oscilloscope. Get the VISA Address from the
' Keysight Connection Expert (installed with Keysight IO Libraries Suite
).
Set myScope.IO = myMgr.Open("<VISA Address>")

' Send a command.
myScope.WriteString ":DISPlay:LABel ON"
```

The ":DISPLAY:LABEL ON" in the above example is called a *program message*. Program messages are explained in more detail in "Program Message Syntax" on page 1293.

Initializing the Interface and the Oscilloscope

To make sure the bus and all appropriate interfaces are in a known state, begin every program with an initialization statement. When using the Keysight VISA COM library, you can use the resource session object's Clear method to clears the interface buffer:

```
Dim myMgr As VisaComLib.ResourceManager
Dim myScope As VisaComLib.FormattedIO488

Set myMgr = New VisaComLib.ResourceManager
Set myScope = New VisaComLib.FormattedIO488

' Open the connection to the oscilloscope. Get the VISA Address from the
' Keysight Connection Expert (installed with Keysight IO Libraries Suite
).
Set myScope.IO = myMgr.Open("<VISA Address>")

' Clear the interface buffer and set the interface timeout to 20 seconds
.
myScope.IO.Clear
myScope.IO.Timeout = 20000
```

After clearing the interface, initialize the instrument to a preset state:

```
myScope.WriteString "*RST"
```

NOTE

Information for Initializing the Instrument

The actual commands and syntax for initializing the instrument are discussed in Chapter 6, "Common (*) Commands," starting on page 179.

Refer to the Keysight IO Libraries Suite documentation for information on initializing the interface.

Using :AUToscale to Automate Oscilloscope Setup

The :AUToscale command performs a very useful function for unknown waveforms by setting up the vertical channel, time base, and trigger level of the instrument.

The syntax for the autoscale command is:

```
myScope.WriteString ":AUToscale"
```

Using Other Oscilloscope Setup Commands

A typical oscilloscope setup would set the vertical range and offset voltage, the horizontal range, delay time, delay reference, trigger mode, trigger level, and slope. An example of the commands that might be sent to the oscilloscope are:

```
myScope.WriteString ":CHANnel1:PROBe 10"
myScope.WriteString ":CHANnel1:RANGe 16"
myScope.WriteString ":CHANnel1:OFFSet 1.00"
myScope.WriteString ":TIMebase:MODE MAIN"
myScope.WriteString ":TIMebase:RANGe 1E-3"
myScope.WriteString ":TIMebase:DELay 100E-6"
```

Vertical is set to 16 V full-scale (2 V/div) with center of screen at 1 V and probe attenuation set to 10. This example sets the time base at 1 ms full-scale (100 ms/div) with a delay of 100 µs.

Example Oscilloscope Setup Code

This program demonstrates the basic command structure used to program the oscilloscope.

```
' Initialize the instrument interface to a known state.
myScope.IO.Clear
myScope.IO.Timeout = 20000   ' Set interface timeout to 20 seconds.

' Initialize the instrument to a preset state.
myScope.WriteString "*RST"

' Set the time base mode to normal with the horizontal time at
' 50 ms/div with 0 s of delay referenced at the center of the
' graticule.
myScope.WriteString ":TIMebase:RANGe 5E-4"   ' Time base to 50 us/div.
myScope.WriteString ":TIMebase:DELay 0"      ' Delay to zero.
myScope.WriteString ":TIMebase:REFerence CENTer"   ' Display ref. at
                                                   ' center.

' Set the vertical range to 1.6 volts full scale with center screen
' at -0.4 volts with 10:1 probe attenuation and DC coupling.
myScope.WriteString ":CHANnel1:PROBe 10"     ' Probe attenuation
                                              ' to 10:1.
myScope.WriteString ":CHANnel1:RANGe 1.6"    ' Vertical range
                                              ' 1.6 V full scale.
myScope.WriteString ":CHANnel1:OFFSet -0.4"  ' Offset to -0.4.
myScope.WriteString ":CHANnel1:COUPling DC"  ' Coupling to DC.

' Configure the instrument to trigger at -0.4 volts with normal
' triggering.
myScope.WriteString ":TRIGger:SWEep NORMal"   ' Normal triggering.
myScope.WriteString ":TRIGger:LEVel -0.4"     ' Trigger level to -0.4.
myScope.WriteString ":TRIGger:SLOPe POSitive" ' Trigger on pos. slope.

' Configure the instrument for normal acquisition.
myScope.WriteString ":ACQuire:TYPE NORMal"   ' Normal acquisition.
```

Capturing Data with the :DIGitize Command

The :DIGitize command captures data that meets the specifications set up by the :ACQuire subsystem. When the digitize process is complete, the acquisition is stopped. The captured data can then be measured by the instrument or transferred to the controller for further analysis. The captured data consists of two parts: the waveform data record, and the preamble.

NOTE

Ensure New Data is Collected

When you change the oscilloscope configuration, the waveform buffers are cleared. Before doing a measurement, send the :DIGitize command to the oscilloscope to ensure new data has been collected.

When you send the :DIGitize command to the oscilloscope, the specified channel signal is digitized with the current :ACQuire parameters. To obtain waveform data, you must specify the :WAVeform parameters for the SOURce channel, the FORMat type, and the number of POINts prior to sending the :WAVeform:DATA? query.

NOTE

Set :TIMebase:MODE to MAIN when using :DIGitize

:TIMebase:MODE must be set to MAIN to perform a :DIGitize command or to perform any :WAVeform subsystem query. A "Settings conflict" error message will be returned if these commands are executed when MODE is set to WINDow (zoomed). Sending the *RST (reset) command will also set the time base mode to normal.

The number of data points comprising a waveform varies according to the number requested in the :ACQuire subsystem. The :ACQuire subsystem determines the number of data points, type of acquisition, and number of averages used by the :DIGitize command. This allows you to specify exactly what the digitized information contains.

The following program example shows a typical setup:

```
myScope.WriteString ":ACQuire:TYPE AVERage"
myScope.WriteString ":ACQuire:COMPlete 100"
myScope.WriteString ":ACQuire:COUNt 8"
myScope.WriteString ":DIGitize CHANnel1"
myScope.WriteString ":WAVeform:SOURce CHANnel1"
myScope.WriteString ":WAVeform:FORMat WORD"
myScope.WriteString ":WAVeform:BYTeorder LSBF"
myScope.WriteString ":WAVeform:POINts 500"
myScope.WriteString ":WAVeform:DATA?"
```

This setup places the instrument into the averaged mode with eight averages. This means that when the :DIGitize command is received, the command will execute until the signal has been averaged at least eight times.

After receiving the :WAVeform:DATA? query, the instrument will start passing the waveform information.

Digitized waveforms are passed from the instrument to the controller by sending a numerical representation of each digitized point. The format of the numerical representation is controlled with the :WAVeform:FORMat command and may be selected as BYTE, WORD, or ASCii.

The easiest method of transferring a digitized waveform depends on data structures, formatting available and I/O capabilities. You must scale the integers to determine the voltage value of each point. These integers are passed starting with the left most point on the instrument's display.

For more information, see the waveform subsystem commands and corresponding program code examples in Chapter 36, ":WAVeform Commands," starting on page 1143.

NOTE

Aborting a Digitize Operation Over the Programming Interface

When using the programming interface, you can abort a digitize operation by sending a Device Clear over the bus (for example, myScope.IO.Clear).

Reading Query Responses from the Oscilloscope

After receiving a query (command header followed by a question mark), 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 interface to the designated listener (typically a controller).

The statement for reading a query response message from an instrument's output queue typically has a format specification for handling the response message.

When using the VISA COM library in Visual Basic, you use different read methods (ReadString, ReadNumber, ReadList, or ReadIEEEBlock) for the various query response formats. For example, to read the result of the query command :CHANnel1:COUPling? you would execute the statements:

```
myScope.WriteString ":CHANnel1:COUPling?"
Dim strQueryResult As String
strQueryResult = myScope.ReadString
```

This reads the current setting for the channel one coupling into the string variable strQueryResult.

All results for queries (sent in one program message) must be read before another program message is sent.

Sending another command before reading the result of the query clears the output buffer and the current response. This also causes an error to be placed in the error queue.

Executing a read statement before sending a query causes the controller to wait indefinitely.

The format specification for handling response messages depends on the programming language.

Reading Query Results into String Variables

The output of the instrument may be numeric or character data depending on what is queried. Refer to the specific command descriptions for the formats and types of data returned from queries.

NOTE

Express String Variables Using Exact Syntax

In Visual Basic, string variables are case sensitive and must be expressed exactly the same each time they are used.

The following example shows numeric data being returned to a string variable:

```
myScope.WriteString ":CHANnel1:RANGe?"
Dim strQueryResult As String
strQueryResult = myScope.ReadString
MsgBox "Range (string):" + strQueryResult
```

After running this program, the controller displays:

Range (string): +40.0E+00

Reading Query Results into Numeric Variables

The following example shows numeric data being returned to a numeric variable:

```
myScope.WriteString ":CHANnel1:RANGe?"
Dim varQueryResult As Variant
varQueryResult = myScope.ReadNumber
MsgBox "Range (variant):" + CStr(varQueryResult)
```

After running this program, the controller displays:

Range (variant): 40

Reading Definite-Length Block Query Response Data

Definite-length block query 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. The 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:

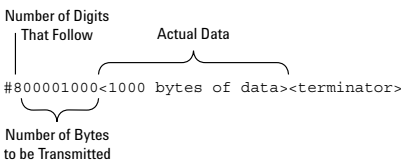


Figure 2 Definite-length block response data

The "8" states the number of digits that follow, and "00001000" states the number of bytes to be transmitted.

The VISA COM library's ReadIEEEBlock and WriteIEEEBlock methods understand the definite-length block syntax, so you can simply use variables that contain the data:

```
' Read oscilloscope setup using ":SYSTem:SETup?" query.
myScope.WriteString ":SYSTem:SETup?"
Dim varQueryResult As Variant
varQueryResult = myScope.ReadIEEEBlock(BinaryType_UI1)

' Write learn string back to oscilloscope using ":SYSTem:SETup" command:
myScope.WriteIEEEBlock ":SYSTem:SETup ", varQueryResult
```

Sending Multiple Queries and Reading Results

You can send multiple queries to the instrument within a single command string, but you must also read them back as a single query result. This can be accomplished by reading them back into a single string variable, multiple string variables, or multiple numeric variables.

For example, to read the :TIMebase:RANGe?;DELay? query result into a single string variable, you could use the commands:

```
myScope.WriteString ":TIMebase:RANGe?;DELay?"
Dim strQueryResult As String
strQueryResult = myScope.ReadString
MsgBox "Timebase range; delay:" + strQueryResult
```

When you read the result of multiple queries into a single string variable, each response is separated by a semicolon. For example, the output of the previous example would be:

```
Timebase range; delay: <range_value>;<delay_value>
```

To read the :TIMebase:RANGe?;DELay? query result into multiple string variables, you could use the ReadList method to read the query results into a string array variable using the commands:

```
myScope.WriteString ":TIMebase:RANGe?;DELay?"
Dim strResults() As String
strResults() = myScope.ReadList(ASCIIType_BSTR)
MsgBox "Timebase range: " + strResults(0) + ", delay: " + strResults(1)
```

To read the :TIMebase:RANGe?;DELay? query result into multiple numeric variables, you could use the ReadList method to read the query results into a variant array variable using the commands:

```
myScope.WriteString ":TIMebase:RANGe?;DELay?"
Dim varResults() As Variant
varResults() = myScope.ReadList
MsgBox "Timebase range: " + FormatNumber(varResults(0) * 1000, 4) + _
       " ms, delay: " + FormatNumber(varResults(1) * 1000000, 4) + " us"
```

See Also • "Multiple Program Message Units" on page 1297

Checking Instrument Status

Status registers track the current status of the instrument. By checking the instrument status, you can find out whether an operation has been completed, whether the instrument is receiving triggers, and more.

For more information, see Chapter 41, “Status Reporting,” starting on page 1253 which explains how to check the status of the instrument.

Other Ways of Sending Commands

Standard Commands for Programmable Instrumentation (SCPI) can also be sent via a Telnet socket or through the browser Instrument IO:

- "Telnet Sockets" on page 69
- "Sending SCPI Commands Using Browser Instrument IO" on page 69

Telnet Sockets

The following information is provided for programmers who wish to control the oscilloscope with SCPI commands in a Telnet session.

To connect to the oscilloscope via a telnet socket, issue the following command:

```
telnet <hostname> 5024
```

where <hostname> is the hostname of the oscilloscope. This will give you a command line with prompt.

For a command line without a prompt, use port 5025. For example:

```
telnet <hostname> 5025
```

Sending SCPI Commands Using Browser Instrument IO

To send SCPI commands using the browser Instrument IO feature, establish a connection to the oscilloscope via LAN as described in the *InfiniiVision HD3-Series Oscilloscopes User's Guide*. When you make the connection to the oscilloscope via LAN and the instrument's welcome page is displayed, select the **Control Instrument** tab, then select **Use Instrument IO**.

4 Sequential (Blocking) vs. Overlapped Commands

IEEE 488.2 makes the distinction between sequential and overlapped commands (and queries):

- *Sequential commands* also known as *blocking commands*, finish their task before the execution of the next command starts.

  These oscilloscope commands and queries are sequential (blocking):

  - :AUToscale
  - :DIGitize
  - :WMEMory<r>:SAVE <source>
  - :WAVeform:DATA?

- *Overlapped commands* run concurrently. Commands following an overlapped command may be started before the overlapped command is completed.

  Some oscilloscope commands are overlapped. For example, the oscilloscope's save and recall commands are overlapped as well as some commands that perform analysis.

With sequential (blocking) commands and queries, the oscilloscope is expected to stop processing inputs, including additional remote commands and queries as well as front panel knobs, until completed.

Pausing Execution Between Overlapped Commands

With overlapped commands, you can use the *OPC? query to prevent any more commands from being executed until the overlapped command is complete. This may be necessary when a command that follows an overlapped command interferes with the overlapped command's processing or analysis. For example:

```
:RECall:SETup "setup.scp";*OPC?;:RECall:ARBitrary "arb_wfm.csv"
```

You can also use the *ESR? query to look at the OPC bit (bit 0) in the Standard Event Status Register to determine when an operation is complete.

Pausing Programs Until Sequential (Blocking) Commands are Complete

Sequential (blocking) commands do not prevent additional commands from being sent to the queue or cause the remote program to wait. For example, if your program does something like:

```
myScope.WriteString ":DIGitize"
Debug.Print "Signal acquired."
```

The "Signal acquired" message will be written immediately after the :DIGitize is sent, not after the acquisition and processing is complete.

To pause the program until a sequential (blocking) command is complete, you must wait for a query result after the sequential (blocking) command. For example, in this case:

```
Public strQueryResult As String
myScope.WriteString ":DIGitize;*OPC?"
strQueryResult = myScope.ReadString
Debug.Print "Signal acquired."
```

The "Signal acquired" message will be written after the acquisition and processing is complete. The *OPC? query is appended to :DIGitize with a semi-colon (;), which essentially ties it to the same thread in the parser. It is immediately dealt with once :DIGitize finishes and gives a "1" back to the program (whether the program uses it or not), allowing the program to move on.

When using a query to wait until a sequential (blocking) command is complete, it is possible for the sequential (blocking) command execution to take longer than the I/O timeout, in which case, there will be a timeout error while waiting for the query results. You can increase the I/O timeout or have your program poll the Status Byte Register using the IO libraries' unblocked ReadSTB function to wait for execution completion.

Using Device Clear to Abort a Sequential (Blocking) Command

When sequential (blocking) commands take too long or fail to complete for some reason, you can send a Device Clear over the bus to clear the input buffer and output queue, reset the parser, and clear any pending commands.

See Also

- "*OPC (Operation Complete)" on page 191
- "*ESR? (Standard Event Status Register)" on page 187
- Chapter 42, “Synchronizing Acquisitions,” starting on page 1269

5 Commands Quick Reference

Command Summary

Table 2 Common (*) Commands Summary

Command	Query	Options and Query Returns
*CLS (see page 184)	n/a	n/a
*ESE <mask> (see page 185)	*ESE? (see page 186)	<mask> ::= 0 to 255; an integer in NR1 format: Bit Weight Name Enables --- ------ ---- ---------- 7 128 PON Power On 6 64 URQ User Request 5 32 CME Command Error 4 16 EXE Execution Error 3 8 DDE Dev. Dependent Error 2 4 QYE Query Error 1 2 RQL Request Control 0 1 OPC Operation Complete
n/a	*ESR? (see page 187)	<status> ::= 0 to 255; an integer in NR1 format
n/a	*IDN? (see page 187)	KEYSIGHT 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
n/a	*LRN? (see page 190)	<learn_string> ::= current instrument setup as a block of data in IEEE 488.2 # format
*OPC (see page 191)	*OPC? (see page 191)	ASCII "1" is placed in the output queue when all pending device operations have completed.

Table 2 Common (*) Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	*OPT? (see page 192)	<return_value> ::= 0,0,<license info> <license info> ::= <All field>, <reserved>, <Enhanced Security>, <Waveform Generator>, <Automotive Serial>, <reserved>, <Embedded Serial>, <reserved>, <reserved>, <reserved>, <reserved>, <reserved>, Memory 20M>, <Memory 50M>, <Memory 100M>, <Bandwidth>, <reserved>, <reserved>, <reserved>, <Demo Partner>, <Demo Keysight>
n/a	*OPT? (see page 192) (cont'd)	<All field> ::= {0 | All(d)} <reserved> ::= 0 <Enhanced Security> ::= {0 | SECA} <Waveform Generator> ::= {0 | WAVEGEN} <Automotive Serial> ::= {0 | HD300AUTA} <Embedded Serial> ::= {0 | HD300EMBA} <Memory 20M> ::= {0 | MEMUP20} <Memory 50M> ::= {0 | MEMUP50} <Memory 100M> ::= {0 | MEMUP100} <Bandwidth> ::= {BW200 | BW350 | BW500 | BW1000} <Demo Partner> ::= {0 | DIS} <Demo Keysight> ::= {0 | D24}
*RCL {<internal_fname> | <external_fname>} (see page 193)	n/a	<internal_fname> ::= quoted ASCII file name string beginning with "/User Files/" <external_fname> ::= quoted ASCII file name string beginning with "/USB1/" (or "/usb/"), "/USB2/", or "/USB3/" depending on the number of USB storage devices connected
*RST (see page 194)	n/a	See *RST (Reset) (see page 194)

Table 2 Common (*) Commands Summary (continued)

Command	Query	Options and Query Returns
*SAV {<internal_fname> | <external_fname>} (see page 197)	n/a	<internal_fname> ::= quoted ASCII file name string beginning with "/User Files/" <external_fname> ::= quoted ASCII file name string beginning with "/USB1/" (or "/usb/"), "/USB2/", or "/USB3/" depending on the number of USB storage devices connected
*SRE <mask> (see page 198)	*SRE? (see page 199)	<mask> ::= sum of all bits that are set, 0 to 255; an integer in NR1 format. <mask> ::= following values: Bit Weight Name Enables --- ------ ---- ---------- 7 128 OPER Operation Status Reg 6 64 ---- (Not used.) 5 32 ESB Event Status Bit 4 16 MAV Message Available 3 8 ---- (Not used.) 2 4 MSG Message 1 2 USR User 0 1 TRG Trigger
n/a	*STB? (see page 200)	<value> ::= 0 to 255; an integer in NR1 format, as shown in the following: Bit Weight Name "1" Indicates --- ------ ---- --------------- 7 128 OPER Operation status condition occurred. 6 64 RQS/ MSS Instrument is requesting service. 5 32 ESB Enabled event status condition occurred. 4 16 MAV Message available. 3 8 ---- (Not used.) 2 4 MSG Message displayed. 1 2 USR User event condition occurred. 0 1 TRG A trigger occurred.
*TRG (see page 203)	n/a	n/a
n/a	*TST? (see page 204)	<result> ::= 0 or non-zero value; an integer in NR1 format
*WAI (see page 205)	n/a	n/a

Table 3 Root (:) Commands Summary

Command	Query	Options and Query Returns
:ACTivity (see page 210)	:ACTivity? (see page 210)	<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? (see page 211)	{0 | 1}; an integer in NR1 format
:AUToscale [<source>[,..,<source>]] (see page 212)	n/a	<source> ::= {CHANnel<n> | DIGital<d> | POD1 | POD2} <source> can be repeated up to 5 times <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:AUToscale:AMODe <value> (see page 214)	:AUToscale:AMODe? (see page 214)	<value> ::= {NORMal | CURRent}}
:AUToscale:CHANnels <value> (see page 215)	:AUToscale:CHANnels? (see page 215)	<value> ::= {ALL | DISPlayed}}
:AUToscale:FDEBug {{0 | OFF} | {1 | ON}} (see page 216)	:AUToscale:FDEBug? (see page 216)	{0 | 1}
:BLANk [<source>] (see page 217)	n/a	<source> ::= {CHANnel<n> | DIGital<d> | POD{1 | 2} | BUS{1 | 2} | FUNCtion<m> | MATH<m> | FFT | SBUS{1 | 2} | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format

Table 3 Root (:) Commands Summary (continued)

Command	Query	Options and Query Returns
:DIGitize [<source>[,..,<source>]] (see page 218)	n/a	<source> ::= {CHANnel<n> | DIGital<d> | POD{1 | 2} | BUS{1 | 2} | FUNCtion<m> | MATH<m> | FFT | SBUS{1 | 2}} <source> can be repeated up to 5 times <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <m> ::= 1 to (# math functions) in NR1 format
n/a	:RSTate? (see page 220)	n/a
:RUN (see page 221)	n/a	n/a
n/a	:SERial? (see page 222)	<return value> ::= quoted string containing serial number
:SINGle (see page 223)	n/a	n/a
n/a	:STATus? <display> (see page 224)	{0 | 1} <display> ::= {CHANnel<n> | DIGital<d> | POD{1 | 2} | BUS{1 | 2} | FUNCtion<m> | MATH<m> | FFT | SBUS{1 | 2} | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format
:STOP (see page 225)	n/a	n/a

Table 3 Root (:) Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:TER? (see page 226)	{0 | 1}
:VIEW <source> (see page 227)	n/a	<source> ::= {CHANnel<n> | DIGital<d> | POD{1 | 2} | BUS{1 | 2} | FUNCtion<m> | MATH<m> | FFT | SBUS{1 | 2} | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format

Table 4 :ACQuire Commands Summary

Command	Query	Options and Query Returns
:ACQuire:BANDwidth <limit> (see page 232)	:ACQuire:BANDwidth? (see page 232)	<limit> ::= {OFF | <global_bw_limit>} in NR3 format
:ACQuire:COMPlete <complete> (see page 233)	:ACQuire:COMPlete? (see page 233)	<complete> ::= 100; an integer in NR1 format
:ACQuire:COUNt <count> (see page 234)	:ACQuire:COUNt? (see page 234)	<count> ::= an integer from 2 to 65536 in NR1 format
:ACQuire:DIGitizer {{0 | OFF} | {1 | ON}} (see page 235)	:ACQuire:DIGitizer? (see page 235)	{0 | 1}
:ACQuire:MANual {{0 | OFF} | {1 | ON}} (see page 236)	:ACQuire:MANual? (see page 236)	{0 | 1}
:ACQuire:MODE <mode> (see page 237)	:ACQuire:MODE? (see page 237)	<mode> ::= {RTIMe | ETIMe | SEGMented}
:ACQuire:POINts[:ANALog] <points> (see page 238)	:ACQuire:POINts[:ANALog]? (see page 238)	<points> ::= {AUTO | <points_value>} <points_value> ::= desired analog memory depth in integer NR1 format

Table 4 :ACQuire Commands Summary (continued)

Command	Query	Options and Query Returns
:ACQuire:POINts[:ANALog]:AUTO {{0 | OFF} | {1 | ON}} (see page 239)	:ACQuire:POINts[:ANALog]:AUTO? (see page 239)	{0 | 1}
:ACQuire:SEGMented:ANALyze (see page 240)	n/a	n/a
:ACQuire:SEGMented:COUNt <count> (see page 241)	:ACQuire:SEGMented:COUNt? (see page 241)	<count> ::= an integer from 2 to 1000 in NR1 format
:ACQuire:SEGMented:INDex <index> (see page 242)	:ACQuire:SEGMented:INDex? (see page 242)	<index> ::= an integer from 1 to 1000 in NR1 format
:ACQuire:SRATe[:ANALog] <rate> (see page 245)	:ACQuire:SRATe[:ANALog]? (see page 245)	<rate> ::= {AUTO | <sample_rate>} <sample_rate> ::= desired analog sample rate in NR3 format
:ACQuire:SRATe[:ANALog]:AUTO {{0 | OFF} | {1 | ON}} (see page 246)	:ACQuire:SRATe[:ANALog]:AUTO? (see page 246)	{0 | 1}
:ACQuire:TYPE <type> (see page 247)	:ACQuire:TYPE? (see page 247)	<type> ::= {NORMal | AVERage | PEAK}

Table 5 :BUS<n> Commands Summary

Command	Query	Options and Query Returns
:BUS<n>:BIT<m> {{0 | OFF} | {1 | ON}} (see page 251)	:BUS<n>:BIT<m>? (see page 251)	{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}} (see page 252)	:BUS<n>:BITS? (see page 252)	<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

Table 5 :BUS<n> Commands Summary (continued)

Command	Query	Options and Query Returns
:BUS<n>:CLEar (see page 254)	n/a	<n> ::= 1 or 2; an integer in NR1 format
:BUS<n>:DISPlay {{0 | OFF} | {1 | ON}} (see page 255)	:BUS<n>:DISPlay? (see page 255)	{0 | 1} <n> ::= 1 or 2; an integer in NR1 format
:BUS<n>:LABel <string> (see page 256)	:BUS<n>:LABel? (see page 256)	<string> ::= quoted ASCII string up to 32 characters <n> ::= 1 or 2; an integer in NR1 format
:BUS<n>:MASK <mask> (see page 257)	:BUS<n>:MASK? (see page 257)	<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

Table 6 :CALibrate Commands Summary

Command	Query	Options and Query Returns
n/a	:CALibrate:DATE? (see page 261)	<return value> ::= <year>,<month>,<day>; all in NR1 format
:CALibrate:LABel <string> (see page 262)	:CALibrate:LABel? (see page 262)	<string> ::= quoted ASCII string up to 32 characters
:CALibrate:OUTPut <signal> (see page 263)	:CALibrate:OUTPut? (see page 263)	<signal> ::= {TRIGgers | MASK | WAVEgen | WGEN1 | WGEN2 | TSOurce} Note: WAVE and WGEN1 are equivalent. Note: WGEN2 only available on models with 2 WaveGen outputs.

Table 6 :CALibrate Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:CALibrate:PROTected? (see page 264)	{"PROTected" | "UNPRotected"}
:CALibrate:STARt (see page 265)	n/a	n/a
n/a	:CALibrate:STATus? (see page 266)	<return value> ::= <status_code>,<status_string> <status_code> ::= an integer status code <status_string> ::= an ASCII status string
n/a	:CALibrate:TEMPerature? (see page 267)	<return value> ::= degrees C delta since last cal in NR3 format
n/a	:CALibrate:TIME? (see page 268)	<return value> ::= <hours>,<minutes>,<seconds>; all in NR1 format

Table 7 :CHANnel<n> Commands Summary

Command	Query	Options and Query Returns
:CHANnel<n>:BWLimit {{0 | OFF} | {1 | ON}} (see page 273)	:CHANnel<n>:BWLimit? (see page 273)	{0 | 1} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:COUPling <coupling> (see page 274)	:CHANnel<n>:COUPling? (see page 274)	<coupling> ::= {AC | DC} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:DISPlay {{0 | OFF} | {1 | ON}} (see page 275)	:CHANnel<n>:DISPlay? (see page 275)	{0 | 1} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:IMPedance <impedance> (see page 276)	:CHANnel<n>:IMPedance? (see page 276)	<impedance> ::= {ONEMeg | FIFTy} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:INVert {{0 | OFF} | {1 | ON}} (see page 277)	:CHANnel<n>:INVert? (see page 277)	{0 | 1} <n> ::= 1 to (# analog channels) in NR1 format

Table 7 :CHANnel<n> Commands Summary (continued)

Command	Query	Options and Query Returns
:CHANnel<n>:LABel <string> (see page 278)	:CHANnel<n>:LABel? (see page 278)	<string> ::= any series of 32 or less ASCII characters enclosed in quotation marks <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:OFFSet <offset>[suffix] (see page 279)	:CHANnel<n>:OFFSet? (see page 279)	<offset> ::= Vertical offset value in NR3 format [suffix] ::= {V | mV} <n> ::= 1-2 or 1-4; in NR1 format
:CHANnel<n>:PROBe <attenuation> (see page 280)	:CHANnel<n>:PROBe? (see page 280)	<attenuation> ::= Probe attenuation ratio in NR3 format <n> ::= 1-2 or 1-4; in NR1 format
:CHANnel<n>:PROBe:BTN <setting> (see page 281)	:CHANnel<n>:PROBe:BTN? (see page 281)	<n> ::= 1 to (# analog channels) in NR1 format <setting> ::= {HEADlight | INFiniimode | RSTop | SINGle | CDISplay | AUToscale | FTRigger | QACTion | NACTion | NONE}
:CHANnel<n>:PROBe:CALibration (see page 282)	n/a	n/a
:CHANnel<n>:PROBe:EXTernal {{0 | OFF} | {1 | ON}} (see page 283)	:CHANnel<n>:PROBe:EXTernal? (see page 283)	<n> ::= 1 to (# analog channels) in NR1 format <setting> ::= {0 | 1}
:CHANnel<n>:PROBe:EXTernal:GAIN <gain_factor> (see page 284)	:CHANnel<n>:PROBe:EXTernal:GAIN? (see page 284)	<n> ::= 1 to (# analog channels) in NR1 format <gain_factor> ::= a real number from 0.0001 to 1000 in NR3 format
:CHANnel<n>:PROBe:EXTernal:UNITs <units> (see page 285)	:CHANnel<n>:PROBe:EXTernal:UNITs? (see page 285)	<n> ::= 1 to (# analog channels) in NR1 format <units> ::= {VOLT | AMPere}
:CHANnel<n>:PROBe:HEAD[:TYPE] <head_param> (see page 286)	:CHANnel<n>:PROBe:HEAD[:TYPE]? (see page 286)	<head_param> ::= {SEND0 | SEND6 | SEND12 | SEND20 | DIFF0 | DIFF6 | DIFF12 | DIFF20 | DSMA | DSMA6 | NONE} <n> ::= 1 to (# analog channels) in NR1 format

Table 7 :CHANnel<n> Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:CHANnel<n>:PROBe:ID? (see page 287)	<probe id> ::= unquoted ASCII string up to 11 characters <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:PROBe:MMODel <value> (see page 288)	:CHANnel<n>:PROBe:MMODel? (see page 288)	<value> ::= {P5205 | P5210 | P6205 | P6241 | P6243 | P6245 | P6246 | P6247 | P6248 | P6249 | P6250 | P6251 | P670X | P671X | TCP202} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:PROBe:MODE <setting> (see page 289)	:CHANnel<n>:PROBe:MODE? (see page 289)	<n> ::= 1 to (# analog channels) in NR1 format <setting> ::= {DIFFerential | DOFFset | SEA | SEB | CM}
:CHANnel<n>:PROBe:RSENse <value> (see page 290)	:CHANnel<n>:PROBe:RSENse? (see page 290)	<value> ::= Ohms in NR3 format <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:PROBe:SKEW <skew_value> (see page 291)	:CHANnel<n>:PROBe:SKEW? (see page 291)	<skew_value> ::= -100 ns to +100 ns in NR3 format <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:PROBe:STYPe <signal type> (see page 292)	:CHANnel<n>:PROBe:STYPe? (see page 292)	<signal type> ::= {DIFFerential | SINGle} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:PROBe:ZOOM {{0 | OFF} | {1 | ON}} (see page 293)	:CHANnel<n>:PROBe:ZOOM? (see page 293)	<setting> ::= {0 | 1} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:PROTection (see page 294)	:CHANnel<n>:PROTection? (see page 294)	{NORM | TRIP} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:RANGe <range>[suffix] (see page 295)	:CHANnel<n>:RANGe? (see page 295)	<range> ::= Vertical full-scale range value in NR3 format [suffix] ::= {V | mV} <n> ::= 1 to (# analog channels) in NR1 format

Table 7 :CHANnel<n> Commands Summary (continued)

Command	Query	Options and Query Returns
:CHANnel<n>:SCALe <scale>[suffix] (see page 296)	:CHANnel<n>:SCALe? (see page 296)	<scale> ::= Vertical units per division value in NR3 format [suffix] ::= {V | mV} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:UNITs <units> (see page 297)	:CHANnel<n>:UNITs? (see page 297)	<units> ::= {VOLT | AMPere} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:VERNier {{0 | OFF} | {1 | ON}} (see page 298)	:CHANnel<n>:VERNier? (see page 298)	{0 | 1} <n> ::= 1 to (# analog channels) in NR1 format

Table 8 :COUNter<c> Commands Summary

Command	Query	Options and Query Returns
n/a	:COUNter<c>:CURRent? (see page 301)	<c> ::= {1 | 2} <value> ::= current counter value in NR3 format
:COUNter<c>:ENABle {{0 | OFF} | {1 | ON}} (see page 302)	:COUNter<c>:ENABle? (see page 302)	<c> ::= {1 | 2} {0 | 1}
:COUNter<c>:MODE <mode> (see page 303)	:COUNter<c>:MODE (see page 303)	<c> ::= {1 | 2} <mode> ::= {FREQuency | PERiod | TOTalize}
:COUNter<c>:NDIGits <value> (see page 304)	:COUNter<c>:NDIGits (see page 304)	<c> ::= {1 | 2} <value> ::= 3 to 8 in NR1 format
:COUNter<c>:SOURce <source> (see page 305)	:COUNter<c>:SOURce? (see page 305)	<c> ::= {1 | 2} <source> ::= {CHANnel<n> | TQEVent} <n> ::= 1 to (# analog channels) in NR1 format
:COUNter<c>:TOTalize:CLEar (see page 306)	n/a	<c> ::= {1 | 2}
:COUNter<c>:TOTalize:SLOPe <slope> (see page 307)	:COUNter<c>:TOTalize:SLOPe? (see page 307)	<c> ::= {1 | 2} <slope> ::= {{NEGative | FALLing} | {POSitive | RISing}}

Table 9 :DIGital<d> Commands Summary

Command	Query	Options and Query Returns
:DIGital:ORDer:ASCending (see page 311)	n/a	n/a
:DIGital:ORDer:DESCending (see page 312)	n/a	n/a
:DIGital:ORDer[:SET] <digital_channels_list> (see page 313)	:DIGital:ORDer[:SET]? (see page 313)	<digital_channels_list> ::= comma-separated list, e.g., DIGital<d>,DIGital<d>,.. <d> ::= 0 to (# digital channels - 1) in NR1 format
:DIGital<d>:DISPlay {{0 | OFF} | {1 | ON}} (see page 314)	:DIGital<d>:DISPlay? (see page 314)	<d> ::= 0 to (# digital channels - 1) in NR1 format {0 | 1}
:DIGital<d>:LABel <string> (see page 315)	:DIGital<d>:LABel? (see page 315)	<d> ::= 0 to (# digital channels - 1) in NR1 format <string> ::= any series of 32 or less ASCII characters enclosed in quotation marks
n/a	:DIGital<d>:SETup? (see page 316)	<d> ::= 0 to (# digital channels - 1) in NR1 format Returns setup information string, for example, ":DIG0:DISP 1;THR +1.40E+00"
:DIGital<d>:SIZE <value> (see page 317)	:DIGital<d>:SIZE? (see page 317)	<d> ::= 0 to (# digital channels - 1) in NR1 format <value> ::= {SMALl | MEDium | LARGe}
:DIGital<d>:THReshold <value>[suffix] (see page 318)	:DIGital<d>:THReshold? (see page 318)	<d> ::= 0 to (# digital channels - 1) 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}

Table 10 :DISPlay Commands Summary

Command	Query	Options and Query Returns
:DISPlay:ANNotation<n> {{0 | OFF} | {1 | ON}} (see page 324)	:DISPlay:ANNotation<n>? (see page 324)	{0 | 1} <n> ::= an integer from 1 to 20 in NR1 format.
:DISPlay:ANNotation<n>:BACKground <mode> (see page 325)	:DISPlay:ANNotation<n>:BACKground? (see page 325)	<mode> ::= {OPAQue | INVerted | TRANsparent} <n> ::= an integer from 1 to 20 in NR1 format.
:DISPlay:ANNotation<n>:COLor <color> (see page 326)	:DISPlay:ANNotation<n>:COLor? (see page 326)	<color> ::= {CH1 | CH2 | CH3 | CH4 | DIG | MATH | REF | MARKer | WHITe | RED} <n> ::= an integer from 1 to 20 in NR1 format.
:DISPlay:ANNotation<n>:GRID <number> (see page 327)	:DISPlay:ANNotation<n>:GRID? (see page 327)	<number> ::= an integer from 1 to 20 in NR1 format. <n> ::= an integer from 1 to 20 in NR1 format.
:DISPlay:ANNotation<n>:MODE <mode> (see page 328)	:DISPlay:ANNotation<n>:MODE? (see page 328)	<mode> ::= {GRID | SOURce} <n> ::= an integer from 1 to 20 in NR1 format.
:DISPlay:ANNotation<n>:SOURce <source> (see page 329)	:DISPlay:ANNotation<n>:SOURce? (see page 329)	<source> ::= {CHANnel<n> | WMEMory<n> | FUNCtion<n> | FFT | DIGital<n>} <n> ::= an integer in NR1 format.
:DISPlay:ANNotation<n>:TEXT <string> (see page 330)	:DISPlay:ANNotation<n>:TEXT? (see page 330)	<string> ::= quoted ASCII string (up to 254 characters) <n> ::= an integer from 1 to 20 in NR1 format.
:DISPlay:ANNotation<n>:XPOSition <value> (see page 331)	:DISPlay:ANNotation<n>:XPOSition? (see page 331)	<value> ::= a decimal from 0.0 to 1.0 in NR3 format that is a percent of the grid width. <n> ::= an integer from 1 to 20 in NR1 format.
:DISPlay:ANNotation<n>:YPOSition <value> (see page 332)	:DISPlay:ANNotation<n>:YPOSition? (see page 332)	<value> ::= a decimal from 0.0 to 1.0 in NR3 format that is a percent of the grid height. <n> ::= an integer from 1 to 20 in NR1 format.

Table 10 :DISPlay Commands Summary (continued)

Command	Query	Options and Query Returns
:DISPlay:BACKlight {{0 | OFF} | {1 | ON}} (see page 333)	n/a	n/a
:DISPlay:CLEar (see page 334)	n/a	n/a
:DISPlay:CLOCk:IGRid {{0 | OFF} | {1 | ON}} (see page 335)	:DISPlay:CLOCk:IGRid? (see page 335)	<setting> ::= {0 | 1}
:DISPlay:CLOCk:IGRid:XPOSition <value> (see page 336)	:DISPlay:CLOCk:IGRid:XPOSition? (see page 336)	<value> ::= a decimal from 0.0 to 1.0 in NR3 format that is a percent of the grid width.
:DISPlay:CLOCk:IGRid:YPOSition <value> (see page 337)	:DISPlay:CLOCk:IGRid:YPOSition? (see page 337)	<value> ::= a decimal from 0.0 to 1.0 in NR3 format that is a percent of the grid height.
:DISPlay:CLOCk[:STATe] {{0 | OFF} | {1 | ON}} (see page 338)	:DISPlay:CLOCk[:STATe]? (see page 338)	<setting> ::= {0 | 1}
n/a	:DISPlay:DATA? [<format>] (see page 339)	<format> ::= {BMP | PNG} <display data> ::= data in IEEE 488.2 # format
:DISPlay:GRATicule:ALABels {{0 | OFF} | {1 | ON}} (see page 340)	:DISPlay:GRATicule:ALABels? (see page 340)	<setting> ::= {0 | 1}
:DISPlay:GRATicule:ALABels:DUAL {{0 | OFF} | {1 | ON}} (see page 341)	:DISPlay:GRATicule:ALABels:DUAL? (see page 341)	<setting> ::= {0 | 1}
:DISPlay:GRATicule:ALABels:IGRid {{0 | OFF} | {1 | ON}} (see page 342)	:DISPlay:GRATicule:ALABels:IGRid? (see page 342)	<setting> ::= {0 | 1}
:DISPlay:GRATicule:COUNt <value> (see page 343)	:DISPlay:GRATicule:COUNt? (see page 343)	<num_grids> ::= an integer from 1 to 4 in NR1 format.
:DISPlay:GRATicule:FSCReen {{0 | OFF} | {1 | ON}} (see page 344)	:DISPlay:GRATicule:FSCReen? (see page 344)	<setting> ::= {0 | 1}
:DISPlay:GRATicule:INTensity <value> (see page 345)	:DISPlay:GRATicule:INTensity? (see page 345)	<value> ::= an integer from 0 to 100 in NR1 format.

Table 10 :DISPlay Commands Summary (continued)

Command	Query	Options and Query Returns
:DISPlay:GRATicule:LAYout <type> (see page 346)	:DISPlay:GRATicule:LAYout? (see page 346)	<type> ::= {STACked | TILed | OVERlay | CUSTom}
:DISPlay:GRATicule:SET <source>,<grid> (see page 347)	n/a	<source> ::= {CHANnel<n> | WMEMory<n> | FUNCtion<n> | FFT | SBUS<n> | GAIN | PHASe} <grid> ::= a grid number integer from 1 to 4 in NR1 format.
n/a	:DISPlay:GRATicule:SOURce? <grid> (see page 348)	<grid> ::= a grid number integer from 1 to 4 in NR1 format. Returns: <sources> ::= comma-separated list of the waveforms displayed in the grid. Can contain items like: CHAN<n>, WMEM<n>, FUNC<n>, FFT, SBUS<n>, GAIN, PHASe
:DISPlay:INTensity:WAVeform <value> (see page 349)	:DISPlay:INTensity:WAVeform? (see page 349)	<value> ::= an integer from 0 to 100 in NR1 format.
:DISPlay:LABel {{0 | OFF} | {1 | ON}} (see page 350)	:DISPlay:LABel? (see page 350)	{0 | 1}
:DISPlay:LABList <binary block> (see page 351)	:DISPlay:LABList? (see page 351)	<binary block> ::= an ordered list of up to 75 labels, each 10 characters maximum, separated by newline characters
:DISPlay:MESSage:CLEar (see page 352)	n/a	n/a
:DISPlay:PERSistence <value> (see page 353)	:DISPlay:PERSistence? (see page 353)	<value> ::= {MINimum | INFinite | <time>} <time> ::= seconds in in NR3 format from 100E-3 to 60E0
:DISPlay:PERSistence:CLEar (see page 354)	n/a	n/a
n/a	:DISPlay:RESults:CATalog? (see page 355)	<windows> ::= list of the results windows displayed from left to right. Can contain: MEAS, MARK, HIST, DVM, COUN, LIST, MTES, FRAN
:DISPlay:RESults:LAYout <format> (see page 356)	:DISPlay:RESults:LAYout? (see page 356)	<format> ::= {TAB | LIST}

Table 10 :DISPlay Commands Summary (continued)

Command	Query	Options and Query Returns
:DISPlay:RESults:LAYout:LIST[:SELect] <results_window> (see page 357)	:DISPlay:RESults:LAYout:LIST[:SELect]? (see page 357)	<results_window> ::= {MEAS | MARK | HIST | DVM | COUN | LIST | MTES | FRAN}
:DISPlay:RESults:LAYout:TAB:LEFT[:SELect] <results_window> (see page 358)	:DISPlay:RESults:LAYout:TAB:LEFT[:SELect]? (see page 358)	<results_window> ::= {MEAS | MARK | HIST | DVM | COUN | LIST | MTES | FRAN}
:DISPlay:RESults:LAYout:TAB:RIGHt[:SELect] <results_window> (see page 359)	:DISPlay:RESults:LAYout:TAB:RIGHt[:SELect]? (see page 359)	<results_window> ::= {MEAS | MARK | HIST | DVM | COUN | LIST | MTES | FRAN}
:DISPlay:RESults:SIZE <size> (see page 360)	:DISPlay:RESults:SIZE? (see page 360)	<size> ::= {CUSTom,<height> | FULL | HIDDen} <height> ::= a pixel height integer from 36 to 1024 in NR1 format.
:DISPlay:TRANsparent {OFF | ON} (see page 361)	:DISPlay:TRANsparent? (see page 361)	{OFF | ON}

Table 11 :DVM Commands Summary

Command	Query	Options and Query Returns
:DVM:ARANge {{0 | OFF} | {1 | ON}} (see page 364)	:DVM:ARANge? (see page 364)	{0 | 1}
n/a	:DVM:CURRent? (see page 365)	<dvm_value> ::= floating-point number in NR3 format
:DVM:ENABle {{0 | OFF} | {1 | ON}} (see page 366)	:DVM:ENABle? (see page 366)	{0 | 1}
n/a	:DVM:FREQuency? (see page 365)	<freq_value> ::= floating-point number in NR3 format
:DVM:MODE <mode> (see page 367)	:DVM:MODE? (see page 367)	<dvm_mode> ::= {ACRMs | DC | DCRMs | FREQuency}
:DVM:SOURce <source> (see page 368)	:DVM:SOURce? (see page 368)	<source> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format

Table 12 :EXTernal Trigger Commands Summary

Command	Query	Options and Query Returns
:EXTernal:BWLimit <bwlimit> (see page 370)	:EXTernal:BWLimit? (see page 370)	<bwlimit> ::= {0 | OFF}
:EXTernal:PROBe <attenuation> (see page 371)	:EXTernal:PROBe? (see page 371)	<attenuation> ::= probe attenuation ratio in NR3 format
:EXTernal:RANGe <range>[<suffix>] (see page 372)	:EXTernal:RANGe? (see page 372)	<range> ::= vertical full-scale range value in NR3 format <suffix> ::= {V | mV}
:EXTernal:UNITs <units> (see page 373)	:EXTernal:UNITs? (see page 373)	<units> ::= {VOLT | AMPere}

Table 13 :FFT Commands Summary

Command	Query	Options and Query Returns
:FFT:AVERage:COUNt <count> (see page 378)	:FFT:AVERage:COUNt? (see page 378)	<count> ::= an integer from 2 to 65536 in NR1 format.
n/a	:FFT:BSIZe? (see page 379)	<bin_size> ::= the FFT resolution as a bin size frequency width in NR3 format.
:FFT:CENTer <frequency> (see page 380)	:FFT:CENTer? (see page 380)	<frequency> ::= the current center frequency in NR3 format. The range of legal values is from -25 GHz to 25 GHz.
:FFT:CLEar (see page 381)	n/a	n/a
:FFT:DETection:POINts <pts_per_span> (see page 382)	:FFT:DETection:POINts? (see page 382)	<pts_per_span> ::= an integer from 640 to 65536 in NR1 format.
:FFT:DETection:TYPE <detection> (see page 383)	:FFT:DETection:TYPE? (see page 383)	<detection> ::= {OFF | SAMPle | PPOSitive | PNEGative | AVERage | NORMal}
:FFT:DISPlay {{0 | OFF} | {1 | ON}} (see page 384)	:FFT:DISPlay? (see page 384)	<s> ::= 1-6, in NR1 format. {0 | 1}
:FFT:DMODe <display_mode> (see page 385)	:FFT:DMODe? (see page 385)	<display_mode> ::= {NORMal | AVERage | MAXHold | MINHold}

Table 13 :FFT Commands Summary (continued)

Command	Query	Options and Query Returns
:FFT:FREQuency:STARt <frequency> (see page 386)	:FFT:FREQuency:STARt? (see page 386)	<frequency> ::= the start frequency in NR3 format.
:FFT:FREQuency:STOP <frequency> (see page 387)	:FFT:FREQuency:STOP? (see page 387)	<frequency> ::= the stop frequency in NR3 format.
:FFT:GATE <gating> (see page 388)	:FFT:GATE? (see page 388)	<gating> ::= {NONE | ZOOM}
:FFT:OFFSet <offset> (see page 389)	:FFT:OFFSet? (see page 389)	<offset> ::= the value at center screen in NR3 format.
:FFT:RANGe <range> (see page 390)	:FFT:RANGe? (see page 390)	<range> ::= the full-scale vertical axis value in NR3 format.
n/a	:FFT:RBWidth? (see page 391)	<resolution_bw> ::= the FFT resolution as a resolution bandwidth frequency in NR3 format.
:FFT:READout <type> (see page 392)	:FFT:READout? (see page 392)	<type> ::= {OFF | SRATe | BSIZe | RBWidth}
:FFT:REFerence <level> (see page 393)	:FFT:REFerence? (see page 393)	<level> ::= the current reference level in NR3 format.
:FFT:SCALe <scale value>[<suffix>] (see page 394)	:FFT:SCALe? (see page 394)	<scale_value> ::= integer in NR1 format. <suffix> ::= dB
:FFT:SOURce <source> (see page 395)	:FFT:SOURce? (see page 395)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m>} <n> ::= 1 to (# analog channels) in NR1 format. <m> ::= 1 to (# math functions) in NR1 format
:FFT:SPAN (see page 396)	:FFT:SPAN? (see page 396)	::= the current frequency span in NR3 format. Legal values are 1 Hz to 100 GHz.
n/a	:FFT:SRATe? (see page 397)	<sample_rate> ::= the FFT resolution as a sample rate frequency in NR3 format.

Table 13 :FFT Commands Summary (continued)

Command	Query	Options and Query Returns
:FFT:VTYPe <units> (see page 398)	:FFT:VTYPe? (see page 398)	<units> ::= {DECibel | VRMS}
:FFT:WINDow <window> (see page 399)	:FFT:WINDow? (see page 399)	<window> ::= {RECTangular | HANNing | FLATtop | BHARris | BARTlett}

Table 14 :FRANalysis Commands Summary

Command	Query	Options and Query Returns
n/a	:FRANalysis:DATA? [SWEep | SINGle] (see page 403)	<binary_block> ::= comma-separated data with newlines at the end of each row
:FRANalysis:ENABle {{0 | OFF} | {1 | ON}} (see page 404)	:FRANalysis:ENABle? (see page 404)	{0 | 1}
:FRANalysis:FREQuency :MODE <setting> (see page 405)	:FRANalysis:FREQuency :MODE? (see page 405)	<setting> ::= {SWEep | SINGle}
:FRANalysis:FREQuency :SINGle <value>[suffix] (see page 406)	:FRANalysis:FREQuency :SINGle? (see page 406)	<value> ::= {20 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000 | 2000000} [suffix] ::= {Hz | kHz| MHz}
:FRANalysis:FREQuency :STARt <value>[suffix] (see page 407)	:FRANalysis:FREQuency :STARt? (see page 407)	<value> ::= {20 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000} [suffix] ::= {Hz | kHz| MHz}
:FRANalysis:FREQuency :STOP <value>[suffix] (see page 408)	:FRANalysis:FREQuency :STOP? (see page 408)	<value> ::= {100 | 1000 | 10000 | 100000 | 1000000 | 10000000 | 20000000} [suffix] ::= {Hz | kHz| MHz}
:FRANalysis:PPDecade <value> (see page 409)	:FRANalysis:PPDecade? (see page 409)	<value> ::= {10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100}
:FRANalysis:RUN (see page 410)	n/a	n/a
:FRANalysis:SOURce:INPut <source> (see page 411)	:FRANalysis:SOURce:INPut? (see page 411)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format

Table 14 :FRANalysis Commands Summary (continued)

Command	Query	Options and Query Returns
:FRANalysis:SOURce:OUTPut <source> (see page 412)	:FRANalysis:SOURce:OUTPut? (see page 412)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:FRANalysis:TRACe <selection> (see page 413)	:FRANalysis:TRACe? (see page 413)	<selection> ::= {NONE | ALL | GAIN | PHASe}[,{GAIN | PHASe}]
:FRANalysis:WGEN:LOAD <impedance> (see page 414)	:FRANalysis:WGEN:LOAD? (see page 414)	<impedance> ::= {ONEMeg | FIFTy}
:FRANalysis:WGEN:VOLTage <amplitude>,[<range>] (see page 415)	:FRANalysis:WGEN:VOLTage? [<range>] (see page 415)	<amplitude> ::= amplitude in volts in NR3 format <range> ::= {F20HZ | F100HZ | F1KHZ | F10KHZ | F100KHZ | F1MHZ | F10MHZ | F20MHZ}
:FRANalysis:WGEN:VOLTage:PROFile {{0 | OFF} | {1 | ON}} (see page 416)	:FRANalysis:WGEN:VOLTage:PROFile? (see page 416)	{0 | 1}

Table 15 :FUNCtion<m> Commands Summary

Command	Query	Options and Query Returns
:FUNCtion<m>:AVERage:COUNt <count> (see page 424)	:FUNCtion<m>:AVERage:COUNt? (see page 424)	<count> ::= an integer from 2 to 65536 in NR1 format <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:BUS:CLOCk <source> (see page 425)	:FUNCtion<m>:BUS:CLOCk? (see page 425)	<source> ::= {CHANnel<n> | DIGital<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:BUS:SLOPe <slope> (see page 426)	:FUNCtion<m>:BUS:SLOPe? (see page 426)	<slope> ::= {NEGative | POSitive | EITHer} <m> ::= 1 to (# math functions) in NR1 format

Table 15 :FUNCtion<m> Commands Summary (continued)

Command	Query	Options and Query Returns
:FUNCtion<m>:BUS:YINCrement <value> (see page 427)	:FUNCtion<m>:BUS:YINCrement? (see page 427)	<value> ::= value per bus code, in NR3 format <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:BUS:YORigin <value> (see page 428)	:FUNCtion<m>:BUS:YORigin? (see page 428)	<value> ::= value at bus code = 0, in NR3 format <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:BUS:YUNits <units> (see page 429)	:FUNCtion<m>:BUS:YUNits? (see page 429)	<units> ::= {VOLT | AMPere | NONE} <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:CLEar (see page 430)	n/a	n/a
:FUNCtion<m>:DISPlay {{0 | OFF} | {1 | ON}} (see page 431)	:FUNCtion<m>:DISPlay? (see page 431)	{0 | 1} <m> ::= 1 to (# math functions) in NR1 format
n/a	:FUNCtion<m>[:FFT]:BSIZe? (see page 432)	<bin_size> ::= Hz in NR3 format <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>[:FFT]:CENTer <frequency> (see page 433)	:FUNCtion<m>[:FFT]:CENTer? (see page 433)	<frequency> ::= the current center frequency in NR3 format. The range of legal values is from -25 GHz to 25 GHz. <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>[:FFT]:DETection:POINts <number_of_buckets> (see page 434)	:FUNCtion<m>[:FFT]:DETection:POINts? (see page 434)	<number_of_buckets> ::= an integer in NR1 format <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>[:FFT]:DETection:TYPE <type> (see page 435)	:FUNCtion<m>[:FFT]:DETection:TYPE? (see page 435)	<type> ::= {OFF | SAMPle | PPOSitive | PNEGative | NORMal | AVERage} <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>[:FFT]:FREQuency:STARt <frequency> (see page 436)	:FUNCtion<m>[:FFT]:FREQuency:STARt? (see page 436)	<frequency> ::= the start frequency in NR3 format. <m> ::= 1 to (# math functions) in NR1 format

Table 15 :FUNCtion<m> Commands Summary (continued)

Command	Query	Options and Query Returns
:FUNCtion<m>[:FFT]:FREQuency:STOP <frequency> (see page 437)	:FUNCtion<m>[:FFT]:FREQuency:STOP? (see page 437)	<frequency> ::= the stop frequency in NR3 format. <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>[:FFT]:GATE <gating> (see page 438)	:FUNCtion<m>[:FFT]:GATE? (see page 438)	<gating> ::= {NONE | ZOOM} <m> ::= 1-4 in NR1 format
:FUNCtion<m>[:FFT]:PHASe:REFerence <ref_point> (see page 439)	:FUNCtion<m>[:FFT]:PHASe:REFerence? (see page 439)	<ref_point> ::= {TRIGger | DISPlay} <m> ::= 1-4 in NR1 format
n/a	:FUNCtion<m>[:FFT]:RBWidth? (see page 440)	<resolution_bw> ::= Hz in NR3 format <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>[:FFT]:READout<n> <readout_type> (see page 441)	:FUNCtion<m>[:FFT]:READout<n>? (see page 441)	<readout_type> ::= {SRATe | BSIZe | RBWidth} <m> ::= 1 to (# math functions) in NR1 format <n> ::= 1-2 in NR1 format, 2 is for dedicated FFT function
:FUNCtion<m>[:FFT]:SPAN (see page 442)	:FUNCtion<m>[:FFT]:SPAN? (see page 442)	::= the current frequency span in NR3 format. Legal values are 1 Hz to 100 GHz. <m> ::= 1 to (# math functions) in NR1 format
n/a	:FUNCtion<m>[:FFT]:SRATe? (see page 443)	<sample_rate> ::= Hz in NR3 format <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>[:FFT]:VTYPe <units> (see page 444)	:FUNCtion<m>[:FFT]:VTYPe? (see page 444)	<units> ::= {DECibel | VRMS} for the FFT (magnitude) operation <units> ::= {DEGRees | RADians} for the FFTPhase operation <m> ::= 1 to (# math functions) in NR1 format

Table 15 :FUNCtion<m> Commands Summary (continued)

Command	Query	Options and Query Returns
:FUNCtion<m>[:FFT]:WINDow <window> (see page 445)	:FUNCtion<m>[:FFT]:WINDow? (see page 445)	<window> ::= {RECTangular | HANNing | FLATtop | BHARris | BARTlett} <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:FREQuency:BANDpass:CENTer <center_freq> (see page 446)	:FUNCtion<m>:FREQuency:BANDpass:CENTer? (see page 446)	<center_freq> ::= center frequency of band-pass filter in NR3 format
:FUNCtion<m>:FREQuency:BANDpass:WIDTh <freq_width> (see page 447)	:FUNCtion<m>:FREQuency:BANDpass:WIDth? (see page 447)	<freq_width> ::= frequency width of band-pass filter in NR3 format
:FUNCtion<m>:FREQuency:HIGHpass <3dB_freq> (see page 448)	:FUNCtion<m>:FREQuency:HIGHpass? (see page 448)	<3dB_freq> ::= 3dB cutoff frequency value in NR3 format <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:FREQuency:LOWPass <3dB_freq> (see page 449)	:FUNCtion<m>:FREQuency:LOWPass? (see page 449)	<3dB_freq> ::= 3dB cutoff frequency value in NR3 format <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:INTegrate:IICondition {{0 | OFF} | {1 | ON}} (see page 450)	:FUNCtion<m>:INTegrate:IICondition? (see page 450)	{0 | 1} <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:INTegrate:IOFFset <input_offset> (see page 451)	:FUNCtion<m>:INTegrate:IOFFset? (see page 451)	<input_offset> ::= DC offset correction in NR3 format. <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:LABel <string> (see page 452)	:FUNCtion<m>:LABel? (see page 452)	<string> ::= quoted ASCII string
:FUNCtion<m>:LINear:GAIN <value> (see page 453)	:FUNCtion<m>:LINear:GAIN? (see page 453)	<value> ::= 'A' in Ax + B, value in NR3 format <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:LINear:OFFSet <value> (see page 454)	:FUNCtion<m>:LINear:OFFSet? (see page 454)	<value> ::= 'B' in Ax + B, value in NR3 format <m> ::= 1 to (# math functions) in NR1 format

Table 15 :FUNCtion<m> Commands Summary (continued)

Command	Query	Options and Query Returns
:FUNCtion<m>:OFFSet <offset> (see page 455)	:FUNCtion<m>:OFFSet? (see page 455)	<offset> ::= the value at center screen in NR3 format. The range of legal values is +/-10 times the current sensitivity of the selected function. <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:OPERation <operation> (see page 456)	:FUNCtion<m>:OPERation? (see page 458)	<operation> ::= {ADD | SUBTract | MULTiply | DIVide | INTegrate | DIFF | FFT | FFTPhase | SQRT | MAGNify | ABSolute | SQUare | LN | LOG | EXP | TEN | LOWPass | HIGHpass | BANDpass | AVERage | LINear | MAXimum | MINimum | MAXHold | MINHold | TRENd} <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:RANGe <range> (see page 460)	:FUNCtion<m>:RANGe? (see page 460)	<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 DIFF function is 80E-3 to 8.0E12 (depends on current sweep speed). The range for the FFT function is 8 to 800 dBV. <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:REFerence <level> (see page 461)	:FUNCtion<m>:REFerence? (see page 461)	<level> ::= the value at center screen in NR3 format. The range of legal values is +/-10 times the current sensitivity of the selected function. <m> ::= 1 to (# math functions) in NR1 format

Table 15 :FUNCtion<m> Commands Summary (continued)

Command	Query	Options and Query Returns
:FUNCtion<m>:SCALe <scale value>[<suffix>] (see page 462)	:FUNCtion<m>:SCALe? (see page 462)	<scale value> ::= integer in NR1 format <suffix> ::= {V | dB} <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:SMOoth:POINts <points> (see page 463)	:FUNCtion<m>:SMOoth:POINts? (see page 463)	<points> ::= odd integer in NR1 format
:FUNCtion<m>:SOURce1 <source> (see page 464)	:FUNCtion<m>:SOURce1? (see page 464)	<source> ::= {CHANnel<n> | FUNCtion<c> | MATH<c> | WMEMory<r> | BUS<b>} <n> ::= 1 to (# analog channels) in NR1 format <c> ::= {1 | 2 | 3}, must be lower than <m> <r> ::= 1 to (# ref waveforms) in NR1 format <b> ::= {1 | 2} <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:SOURce2 <source> (see page 466)	:FUNCtion<m>:SOURce2? (see page 466)	<source> ::= {CHANnel<n> | WMEMory<r> | NONE} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:TRENd:NMEasurement MEAS<n> (see page 467)	:FUNCtion<m>:TRENd:NMEasurement? (see page 467)	<n> ::= # of installed measurement, from 1 to 8 <m> ::= 1 to (# math functions) in NR1 format

Table 16 :HCOPy Commands Summary

Command	Query	Options and Query Returns
n/a	:HCOPy:SDUMp:DATA? (see page 470)	<display_data> ::= binary block data in IEEE-488.2 # format.
:HCOPy:SDUMp[:DATA]:FORMat <format> (see page 471)	:HCOPy:SDUMp[:DATA]:FORMat? (see page 471)	<format> ::= {BMP | PNG}

Table 17 :HISTogram Commands Summary

Command	Query	Options and Query Returns
:HISTogram:AXIS <axis> (see page 476)	:HISTogram:AXIS? (see page 476)	<axis> ::= {VERTical | HORizontal}
:HISTogram:DISPlay {{0 | OFF} | {1 | ON}} (see page 477)	:HISTogram:DISPlay? (see page 477)	{0 | 1}
:HISTogram:MEASurement MEAS<n> (see page 478)	:HISTogram:MEASurement? (see page 478)	<n> ::= # of installed measurement, from 1 to 10
:HISTogram:MODE <mode> (see page 479)	:HISTogram:MODE? (see page 479)	<mode> ::= {OFF | WAVeform | MEASurement}
:HISTogram:RESet (see page 480)	n/a	n/a
:HISTogram:SIZE <size> (see page 481)	:HISTogram:SIZE? (see page 481)	<size> ::= 1.0 to 5.0 for vertical histograms, 1.0 to 4.0 for horizontal histograms, in NR3 format
:HISTogram:TYPE <type> (see page 482)	:HISTogram:TYPE? (see page 482)	<type> ::= {VERTical | HORizontal | MEASurement}
:HISTogram:WINDow:BLIMit <limit> (see page 483)	:HISTogram:WINDow:BLIMit? (see page 483)	<limit> ::= bottom value in NR3 format
:HISTogram:WINDow:LLIMit <limit> (see page 484)	:HISTogram:WINDow:LLIMit? (see page 484)	<limit> ::= left value in NR3 format
:HISTogram:WINDow:RLIMit <limit> (see page 485)	:HISTogram:WINDow:RLIMit? (see page 485)	<limit> ::= right value in NR3 format

Table 17 :HISTogram Commands Summary (continued)

Command	Query	Options and Query Returns
:HISTogram:WINDow:SOURce <source> (see page 486)	:HISTogram:WINDow:SOURce? (see page 486)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format
:HISTogram:WINDow:TLIMit <limit> (see page 487)	:HISTogram:WINDow:TLIMit? (see page 487)	<limit> ::= top value in NR3 format

Table 18 :LISTer Commands Summary

Command	Query	Options and Query Returns
n/a	:LISTer:DATA? (see page 490)	<binary_block> ::= comma-separated data with newlines at the end of each row
:LISTer:DISPlay {{OFF | 0} | {SBUS1 | ON | 1} | {SBUS2 | 2}} (see page 491)	:LISTer:DISPlay? (see page 491)	{OFF | SBUS1 | SBUS2}
:LISTer:REFerence <time_ref> (see page 492)	:LISTer:REFerence? (see page 492)	<time_ref> ::= {TRIGger | PREVious}

Table 19 :LTESt Commands Summary

Command	Query	Options and Query Returns
:LTESt:COPY (see page 495)	n/a	n/a
:LTESt:COPY:ALL (see page 496)	n/a	n/a
:LTESt:COPY:MARGin <percent> (see page 497)	:LTESt:COPY:MARGin? (see page 497)	<percent> ::= copy margin percent in NR1 format
:LTESt:ENABle {{0 | OFF} | {1 | ON}} (see page 498)	:LTESt:ENABle? (see page 498)	<setting> ::= {0 | 1}

Table 19 :LTESt Commands Summary (continued)

Command	Query	Options and Query Returns
:LTESt:FAIL <condition> (see page 499)	:LTESt:FAIL? (see page 499)	<condition> ::= {INSide | OUTSide}
:LTESt:LLIMit <lower_value> (see page 500)	:LTESt:LLIMit? (see page 500)	<lower_value> ::= a real number in NR3 format
:LTESt:MEASurement {MEAS<n>} (see page 501)	:LTESt:MEASurement? (see page 501)	<n> ::= # of installed measurement, from 1 to 10
n/a	:LTESt:RESults? [{MEAS<n>}] (see page 502)	<n> ::= # of installed measurement, from 1 to 10
:LTESt:RUMode:SOFailure {{0 | OFF} | {1 | ON}} (see page 503)	:LTESt:RUMode:SOFailure? (see page 503)	<setting> ::= {0 | 1}
:LTESt:TEST {{0 | OFF} | {1 | ON}} (see page 504)	:LTESt:TEST? (see page 504)	<setting> ::= {0 | 1}
:LTESt:ULIMit <upper_value> (see page 505)	:LTESt:ULIMit? (see page 505)	<upper_value> ::= a real number in NR3 format

Table 20 :MARKer Commands Summary

Command	Query	Options and Query Returns
n/a	:MARKer:DYDX? (see page 510)	<return_value> ::= ΔY/ΔX value in NR3 format
:MARKer:MODE <mode> (see page 511)	:MARKer:MODE? (see page 511)	<mode> ::= {OFF | MEASurement | MANual | WAVeform | BINary | HEX}
:MARKer:X1:DISPlay {{0 | OFF} | {1 | ON}} (see page 512)	:MARKer:X1:DISPlay? (see page 512)	<setting> ::= {0 | 1}
:MARKer:X1Position <position>[suffix] (see page 513)	:MARKer:X1Position? (see page 513)	<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

Table 20 :MARKer Commands Summary (continued)

Command	Query	Options and Query Returns
:MARKer:X1Y1source <source> (see page 514)	:MARKer:X1Y1source? (see page 514)	<source> ::= {CHANnel<n> | FUNCtion<m> | FFT | MATH<m> | WMEMory<r> | HISTogram} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= <source>
:MARKer:X2:DISPlay {{0 | OFF} | {1 | ON}} (see page 515)	:MARKer:X2:DISPlay? (see page 515)	<setting> ::= {0 | 1}
:MARKer:X2Position <position>[suffix] (see page 516)	:MARKer:X2Position? (see page 516)	<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> (see page 517)	:MARKer:X2Y2source? (see page 517)	<source> ::= {CHANnel<n> | FUNCtion<m> | FFT | MATH<m> | WMEMory<r> | HISTogram} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= <source>
n/a	:MARKer:XDELta? (see page 518)	<return_value> ::= X cursors delta value in NR3 format
:MARKer:XUNits <mode> (see page 519)	:MARKer:XUNits? (see page 519)	<units> ::= {SEConds | HERTz | DEGRees | PERCent}
:MARKer:XUNits:USE (see page 520)	n/a	n/a
:MARKer:Y1:DISPlay {{0 | OFF} | {1 | ON}} (see page 521)	:MARKer:Y1:DISPlay? (see page 521)	<setting> ::= {0 | 1}

Table 20 :MARKer Commands Summary (continued)

Command	Query	Options and Query Returns
:MARKer:Y1Position <position>[suffix] (see page 522)	:MARKer:Y1Position? (see page 522)	<position> ::= Y1 cursor position value in NR3 format [suffix] ::= {V | mV | dB} <return_value> ::= Y1 cursor position value in NR3 format
:MARKer:Y2:DISPlay {{0 | OFF} | {1 | ON}} (see page 523)	:MARKer:Y2:DISPlay? (see page 523)	<setting> ::= {0 | 1}
:MARKer:Y2Position <position>[suffix] (see page 524)	:MARKer:Y2Position? (see page 524)	<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? (see page 525)	<return_value> ::= Y cursors delta value in NR3 format
:MARKer:YUNits <mode> (see page 526)	:MARKer:YUNits? (see page 526)	<units> ::= {BASE | PERCent}
:MARKer:YUNits:USE (see page 527)	n/a	n/a

Table 21 :MEASure Commands Summary

Command	Query	Options and Query Returns
:MEASure:AREa [<interval>][,<source>] (see page 547)	:MEASure:AREa? [<interval>][,<source>] (see page 547)	<interval> ::= {CYCLe | DISPlay} <source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= area in volt-seconds, NR3 format
:MEASure:BRATe [<source>] (see page 548)	:MEASure:BRATe? [<source>] (see page 548)	<source> ::= {<digital channels> | CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <digital channels> ::= DIGital<d> <d> ::= 0 to (# digital channels - 1) in NR1 format <n> ::= 1 to (# of analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= bit rate in Hz, NR3 format
:MEASure:BWIDth [<source>] (see page 549)	:MEASure:BWIDth? [<source>] (see page 549)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= burst width in seconds, NR3 format
:MEASure:CLEar (see page 550)	n/a	n/a

Table 21 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:COUNter [<source>] (see page 551)	:MEASure:COUNter? [<source>] (see page 551)	<source> ::= {CHANnel<n> | DIGital<d> | EXTernal} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= counter frequency in Hertz in NR3 format
:MEASure:DELay [<source1>] [,<source2>] (see page 553)	:MEASure:DELay? [<source1>] [,<source2>] (see page 553)	<source1,2> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r> | <digital channels>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <digital channels> ::= DIGital<d> <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= floating-point number delay time in seconds in NR3 format
:MEASure:DELay:DEFine <source1_edge_slope>, <source1_edge_number>, <source1_edge_threshold>, <source2_edge_slope>, <source2_edge_number>, <source2_edge_threshold> (see page 555)	:MEASure:DELay:DEFine? (see page 555)	<source1_edge_slope>, <source2_edge_slope> ::= {RISing | FALLing} <source1_edge_number>, <source2_edge_number> ::= 0 to 1000 in NR1 format <source1_edge_threshold>, <source2_edge_threshold> ::= {LOWer | MIDDle | UPPer}

Table 21 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:DUTYcycle [<source>] (see page 557)	:MEASure:DUTYcycle? [<source>] (see page 557)	<source> ::= {CHANnel<n> | DIGital<d> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= ratio of positive pulse width to period in NR3 format
:MEASure:FALLtime [<source>] (see page 558)	:MEASure:FALLtime? [<source>] (see page 558)	<source> ::= {CHANnel<n> | DIGital<d> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= time in seconds between the lower and upper thresholds in NR3 format

Table 21 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:FFT:ACPR <chan_width>, <chan_spacing>, <chan>[,<source>] (see page 559)	:MEASure:FFT:ACPR? <chan_width>, <chan_spacing>, <chan>[,<source>] (see page 559)	<chan_width> ::= width of main range and sideband channels, Hz in NR3 format <chan_spacing> ::= spacing between main range and sideband channels, Hz in NR3 format <chan> ::= {CENTer | HIGH<sb> | LOW<sb>} <sb> ::= sideband 1 to 5 <source> ::= {FUNCtion<m> | MATH<m> | FFT} (source must be an FFT waveform) <m> ::= 1 to (# math functions) in NR1 format <return_value> ::= adjacent channel power ratio, dBV in NR3 format
:MEASure:FFT:CPOWer [<source>] (see page 560)	:MEASure:FFT:CPOWer? [<source>] (see page 560)	<source> ::= {FUNCtion<m> | MATH<m> | FFT} (source must be an FFT waveform) <m> ::= 1 to (# math functions) in NR1 format <return_value> ::= spectral channel power, dBV in NR3 format
:MEASure:FFT:OBW <percentage>[,<source>] (see page 561)	:MEASure:FFT:OBW? <percentage>[,<source>] (see page 561)	<percentage> ::= percent of spectral power occupied bandwidth is measured for in NR3 format <source> ::= {FUNCtion<m> | MATH<m> | FFT} (source must be an FFT waveform) <m> ::= 1 to (# math functions) in NR1 format <return_value> ::= occupied bandwidth, Hz in NR3 format

Table 21 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:FFT:THD [<source>] (see page 562)	:MEASure:FFT:THD? [<source>] (see page 562)	<source> ::= {FUNCtion<m> | MATH<m> | FFT} (source must be an FFT waveform) <m> ::= 1 to (# math functions) in NR1 format <return_value> ::= total harmonic distortion ratio percent in NR3 format
:MEASure:FREQuency [<source>] (see page 563)	:MEASure:FREQuency? [<source>] (see page 563)	<source> ::= {CHANnel<n> | DIGital<d> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= frequency in Hertz in NR3 format
n/a	:MEASure:HISTogram:BWIDth? (see page 564)	<value> ::= bin width in NR3 format
n/a	:MEASure:HISTogram:HITS? (see page 565)	<value> ::= # of total hits in NR3 format
n/a	:MEASure:HISTogram:M1S? (see page 566)	<value> ::= % of hits within +/-1 std dev in NR3 format
n/a	:MEASure:HISTogram:M2S? (see page 567)	<value> ::= % of hits within +/-2 std devs in NR3 format
n/a	:MEASure:HISTogram:M3S? (see page 568)	<value> ::= % of hits within +/-3 std devs in NR3 format
n/a	:MEASure:HISTogram:MAXimum? (see page 569)	<value> ::= value of max bin in NR3 format
n/a	:MEASure:HISTogram:MEAN? (see page 570)	<value> ::= mean of histogram in NR3 format
n/a	:MEASure:HISTogram:MEDian? (see page 571)	<value> ::= median of histogram in NR3 format
n/a	:MEASure:HISTogram:MINimum? (see page 572)	<value> ::= value of min bin in NR3 format

Table 21 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:MEASure:HISTogram:MODE? (see page 573)	<value> ::= mode of histogram in NR3 format
n/a	:MEASure:HISTogram:PEAK? (see page 574)	<value> ::= # of hits in max bin in NR3 format
n/a	:MEASure:HISTogram:PPEak? (see page 575)	<value> ::= delta value between max bin and min bin in NR3 format
n/a	:MEASure:HISTogram:SDEViation? (see page 576)	<value> ::= standard deviation of histogram in NR3 format
:MEASure:NDUTy [<source>] (see page 577)	:MEASure:NDUTy? [<source>] (see page 577)	<source> ::= {CHANnel<n> | DIGital<d> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= ratio of negative pulse width to period in NR3 format
:MEASure:NEDGes [<source>] (see page 578)	:MEASure:NEDGes? [<source>] (see page 578)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the falling edge count in NR3 format

Table 21 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:NPULses [<source>] (see page 579)	:MEASure:NPULses? [<source>] (see page 579)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r> | <digital channels>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <digital channels> ::= DIGital<d> <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= the falling pulse count in NR3 format
:MEASure:NWIDth [<source>] (see page 580)	:MEASure:NWIDth? [<source>] (see page 580)	<source> ::= {CHANnel<n> | DIGital<d> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= negative pulse width in seconds-NR3 format
:MEASure:OVERshoot [<source>] (see page 581)	:MEASure:OVERshoot? [<source>] (see page 581)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the percent of the overshoot of the selected waveform in NR3 format

Table 21 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:PEDGes [<source>] (see page 583)	:MEASure:PEDGes? [<source>] (see page 583)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the rising edge count in NR3 format
:MEASure:PERiod [<source>] (see page 584)	:MEASure:PERiod? [<source>] (see page 584)	<source> ::= {CHANnel<n> | DIGital<d> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= waveform period in seconds in NR3 format
:MEASure:PHASe [<source1>] [,<source2>] (see page 585)	:MEASure:PHASe? [<source1>] [,<source2>] (see page 585)	<source1,2> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the phase angle value in degrees in NR3 format

Table 21 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:PPULses [<source>] (see page 586)	:MEASure:PPULses? [<source>] (see page 586)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r> | <digital channels>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <digital channels> ::= DIGital<d> <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= the rising pulse count in NR3 format
:MEASure:PREShoot [<source>] (see page 587)	:MEASure:PREShoot? [<source>] (see page 587)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the percent of preshoot of the selected waveform in NR3 format
:MEASure:PWIDth [<source>] (see page 588)	:MEASure:PWIDth? [<source>] (see page 588)	<source> ::= {CHANnel<n> | DIGital<d> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= width of positive pulse in seconds in NR3 format

Table 21 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:QUICk <source> (see page 589)	n/a	<source> ::= {CHANnel<n>}
n/a	:MEASure:RESults? <result_list> (see page 590)	<result_list> ::= comma-separated list of measurement results
:MEASure:RISetime [<source>] (see page 594)	:MEASure:RISetime? [<source>] (see page 594)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= rise time in seconds in NR3 format
:MEASure:SDEViation [<source>] (see page 595)	:MEASure:SDEViation? [<source>] (see page 595)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= calculated std deviation in NR3 format
:MEASure:SHOW {{0 | OFF} | {1 | ON}} (see page 596)	:MEASure:SHOW? (see page 596)	{0 | 1}
:MEASure:SLEWrate [<source>[,<slope]] (see page 597)	:MEASure:SLEWrate? [<source>[,<slope]] (see page 597)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# of analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format

Table 21 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:SOURce <source1> [,<source2>] (see page 599)	:MEASure:SOURce? (see page 599)	<source1,2> ::= {CHANnel<n> | DIGital<d> | FUNCtion<m> | MATH<m> | WMEMory<r> | EXTernal} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= {<source> | NONE}
:MEASure:STATistics <type> (see page 602)	:MEASure:STATistics? (see page 602)	<type> ::= {{ON | 1} | CURRent | MEAN | MINimum | MAXimum | STDDev | COUNt} ON ::= all statistics returned
:MEASure:STATistics:DISPlay {{0 | OFF} | {1 | ON}} (see page 603)	:MEASure:STATistics:DISPlay? (see page 603)	{0 | 1}
:MEASure:STATistics:INCRement (see page 604)	n/a	n/a
:MEASure:STATistics:MCOunt <setting> (see page 605)	:MEASure:STATistics:MCOunt? (see page 605)	<setting> ::= {INFinite | <count>} <count> ::= 2 to 2000 in NR1 format
:MEASure:STATistics:RESet (see page 606)	n/a	n/a
:MEASure:STATistics:RSDeviation {{0 | OFF} | {1 | ON}} (see page 607)	:MEASure:STATistics:RSDeviation? (see page 607)	{0 | 1}

Table 21 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:TEDGe [<slope>,]<occurrence>[,<source>] (see page 608)	:MEASure:TEDGe? [<slope>,]<occurrence>[,<source>] (see page 609)	<slope> ::= {RISing | FALLing | BOTH} <occurrence> ::= [+ | -]<number> <number> ::= the edge number in NR1 format <source> ::= {<digital channels> | CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <digital channels> ::= DIGital<d> <n> ::= 1 to (# of analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= time in seconds of the specified transition
:MEASure:THResholds:ABSolute <upper>,<middle>,<lower> (see page 612)	:MEASure:THResholds:ABSolute? (see page 612)	<upper>,<middle>,<lower> ::= A number specifying the upper, middle, and lower threshold absolute values in NR3 format.
:MEASure:THResholds:METHod <threshold_mode> (see page 613)	:MEASure:THResholds:METHod? (see page 613)	<threshold_mode> ::= {PERCent | ABSolute}
:MEASure:THResholds:PERCent <upper>,<middle>,<lower> (see page 614)	:MEASure:THResholds:PERCent? (see page 614)	<upper>,<middle>,<lower> ::= A number specifying the upper, middle, and lower threshold percentage values between Vbase and Vtop in NR3 format.
:MEASure:THResholds:STANdard (see page 615)	n/a	n/a

Table 21 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:MEASure:TVALue? <value>, [<slope>]<occurrence> [,<source>] (see page 616)	<value> ::= voltage level that the waveform must cross. <slope> ::= direction of the waveform when <value> is crossed. <occurrence> ::= transitions reported. <source> ::= {CHANnel<n> | DIGital<d> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= time in seconds of specified voltage crossing in NR3 format
:MEASure:VAMPlitude [<source>] (see page 618)	:MEASure:VAMPlitude? [<source>] (see page 618)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the amplitude of the selected waveform in volts in NR3 format

Table 21 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:VAVerage [<interval>][,<source>] (see page 619)	:MEASure:VAVerage? [<interval>][,<source>] (see page 619)	<interval> ::= {CYCLe | DISPlay} <source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | FFT | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= calculated average voltage in NR3 format
:MEASure:VBASe [<source>] (see page 620)	:MEASure:VBASe? [<source>] (see page 620)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <base_voltage> ::= voltage at the base of the selected waveform in NR3 format
:MEASure:VMAX [<source>] (see page 621)	:MEASure:VMAX? [<source>] (see page 621)	<source> ::= {CHANnel<n> | FUNCtion<m> | FFT | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= maximum voltage of the selected waveform in NR3 format

Table 21 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:VMIN [<source>] (see page 622)	:MEASure:VMIN? [<source>] (see page 622)	<source> ::= {CHANnel<n> | FUNCtion<m> | FFT | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= minimum voltage of the selected waveform in NR3 format
:MEASure:VPP [<source>] (see page 623)	:MEASure:VPP? [<source>] (see page 623)	<source> ::= {CHANnel<n> | FUNCtion<m> | FFT | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= voltage peak-to-peak of the selected waveform in NR3 format
:MEASure:VRATio [<interval>][,<source 1>] [,<source2>] (see page 624)	:MEASure:VRATio? [<interval>][,<source 1>] [,<source2>] (see page 624)	<interval> ::= {CYCLe | DISPlay} <source1,2> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the ratio value in dB in NR3 format

Table 21 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:VRMS [<interval>] [,<type>] [,<source>] (see page 625)	:MEASure:VRMS? [<interval>] [,<type>] [,<source>] (see page 625)	<interval> ::= {CYCLe | DISPlay} <type> ::= {AC | DC} <source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= calculated dc RMS voltage in NR3 format
:MEASure:VTOP [<source>] (see page 626)	:MEASure:VTOP? [<source>] (see page 626)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= voltage at the top of the waveform in NR3 format
:MEASure:WINDow <type> (see page 627)	:MEASure:WINDow? (see page 627)	<type> ::= {MAIN | ZOOM | AUTO | GATE}
:MEASure:XMAX [<source>] (see page 628)	:MEASure:XMAX? [<source>] (see page 628)	<source> ::= {CHANnel<n> | FUNCtion<m> | FFT | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= horizontal value of the maximum in NR3 format

Table 21 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:XMIN [<source>] (see page 629)	:MEASure:XMIN? [<source>] (see page 629)	<source> ::= {CHANnel<n> | FUNCtion<m> | FFT | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= horizontal value of the minimum in NR3 format
:MEASure:YATX <horiz_location>[,<source>] (see page 630)	:MEASure:YATX? <horiz_location>[,<source>] (see page 630)	<horiz_location> ::= displayed time from trigger in seconds in NR3 format <source> ::= {CHANnel<n> | DIGital<d> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= voltage at the specified time in NR3 format

Table 22 :MTESt Commands Summary

Command	Query	Options and Query Returns
:MTESt:ALL {{0 | OFF} | {1 | ON}} (see page 637)	:MTESt:ALL? (see page 637)	{0 | 1}
:MTESt:AMASk:CREate (see page 638)	n/a	n/a

Table 22 :MTESt Commands Summary (continued)

Command	Query	Options and Query Returns
:MTESt:AMASk:SOURce <source> (see page 639)	:MTESt:AMASk:SOURce? (see page 639)	<source> ::= CHANnel<n> <n> ::= {1 | 2 | 3 | 4} for 4ch models <n> ::= {1 | 2} for 2ch models
:MTESt:AMASk:UNITs <units> (see page 640)	:MTESt:AMASk:UNITs? (see page 640)	<units> ::= {CURRent | DIVisions}
:MTESt:AMASk:XDELta <value> (see page 641)	:MTESt:AMASk:XDELta? (see page 641)	<value> ::= X delta value in NR3 format
:MTESt:AMASk:YDELta <value> (see page 642)	:MTESt:AMASk:YDELta? (see page 642)	<value> ::= Y delta value in NR3 format
n/a	:MTESt:COUNt:FWAVeforms? [CHANnel<n>] (see page 643)	<failed> ::= number of failed waveforms in NR1 format
:MTESt:COUNt:RESet (see page 644)	n/a	n/a
n/a	:MTESt:COUNt:TIME? (see page 645)	<time> ::= elapsed seconds in NR3 format
n/a	:MTESt:COUNt:WAVeforms? (see page 646)	<count> ::= number of waveforms in NR1 format
:MTESt:DATA <mask> (see page 647)	:MTESt:DATA? (see page 647)	<mask> ::= data in IEEE 488.2 # format.
:MTESt:DELete (see page 648)	n/a	n/a
:MTESt:ENABle {{0 | OFF} | {1 | ON}} (see page 649)	:MTESt:ENABle? (see page 649)	{0 | 1}
:MTESt:LOCK {{0 | OFF} | {1 | ON}} (see page 650)	:MTESt:LOCK? (see page 650)	{0 | 1}
:MTESt:RMODe <rmode> (see page 651)	:MTESt:RMODe? (see page 651)	<rmode> ::= {FOREver | TIME | SIGMa | WAVeforms}
:MTESt:RMODe:FACTion:MEASure {{0 | OFF} | {1 | ON}} (see page 652)	:MTESt:RMODe:FACTion:MEASure? (see page 652)	{0 | 1}
:MTESt:RMODe:FACTion:PRINt {{0 | OFF} | {1 | ON}} (see page 653)	:MTESt:RMODe:FACTion:PRINt? (see page 653)	{0 | 1}

Table 22 :MTESt Commands Summary (continued)

Command	Query	Options and Query Returns
:MTESt:RMODe:FACTion:SAVE {{0 | OFF} | {1 | ON}} (see page 654)	:MTESt:RMODe:FACTion:SAVE? (see page 654)	{0 | 1}
:MTESt:RMODe:FACTion:STOP {{0 | OFF} | {1 | ON}} (see page 655)	:MTESt:RMODe:FACTion:STOP? (see page 655)	{0 | 1}
:MTESt:RMODe:SIGMa <level> (see page 656)	:MTESt:RMODe:SIGMa? (see page 656)	<level> ::= from 0.1 to 9.3 in NR3 format
:MTESt:RMODe:TIME <seconds> (see page 657)	:MTESt:RMODe:TIME? (see page 657)	<seconds> ::= from 1 to 86400 in NR3 format
:MTESt:RMODe:WAVeforms <count> (see page 658)	:MTESt:RMODe:WAVeforms? (see page 658)	<count> ::= number of waveforms in NR1 format
:MTESt:SCALe:BIND {{0 | OFF} | {1 | ON}} (see page 659)	:MTESt:SCALe:BIND? (see page 659)	{0 | 1}
:MTESt:SCALe:X1 <x1_value> (see page 660)	:MTESt:SCALe:X1? (see page 660)	<x1_value> ::= X1 value in NR3 format
:MTESt:SCALe:XDELta <xdelta_value> (see page 661)	:MTESt:SCALe:XDELta? (see page 661)	<xdelta_value> ::= X delta value in NR3 format
:MTESt:SCALe:Y1 <y1_value> (see page 662)	:MTESt:SCALe:Y1? (see page 662)	<y1_value> ::= Y1 value in NR3 format
:MTESt:SCALe:Y2 <y2_value> (see page 663)	:MTESt:SCALe:Y2? (see page 663)	<y2_value> ::= Y2 value in NR3 format
:MTESt:SOURce <source> (see page 664)	:MTESt:SOURce? (see page 664)	<source> ::= {CHANnel<n> | NONE} <n> ::= {1 | 2 | 3 | 4} for 4ch models <n> ::= {1 | 2} for 2ch models
n/a	:MTESt:TITLe? (see page 665)	<title> ::= a quoted string of up to 128 ASCII characters

Table 23 :POD<n> Commands Summary

Command	Query	Options and Query Returns
:POD<n>:DISPlay {{0 | OFF} | {1 | ON}} (see page 669)	:POD<n>:DISPlay? (see page 669)	{0 | 1} <n> ::= 1-2 in NR1 format
:POD<n>:NIBBle<n>:THReshold <value> (see page 670)	:POD<n>:NIBBle<n>:THReshold? (see page 670)	<n> ::= 1-2 in NR1 format <value> :: = {CMOS | ECL | TTL | <decimal>[<suffix>]} <decimal> ::= -8.00 to +8.00 in NR3 format [suffix] ::= {V | mV | uV }
:POD<n>:SIZE <value> (see page 672)	:POD<n>:SIZE? (see page 672)	<value> ::= {SMALl | MEDium | LARGe}
:POD<n>:THReshold <value> (see page 673)	:POD<n>:THReshold? (see page 673)	<n> ::= 1-2 in NR1 format <value> :: = {CMOS | ECL | TTL | <decimal>[<suffix>]} <decimal> ::= -8.00 to +8.00 in NR3 format [suffix] ::= {V | mV | uV }

Table 24 :RECall Commands Summary

Command	Query	Options and Query Returns
:RECall:ARBitrary:[STARt] [{<internal_fname> | <external_fname>}][, <column>][, <wavegen_id>] (see page 678)	n/a	<internal_fname> ::= quoted ASCII file name string beginning with "/User Files/" <external_fname> ::= quoted ASCII file name string beginning with "/USB1/" (or "/usb/"), "/USB2/", or "/USB3/" depending on the number of USB storage devices connected <column> ::= Column in CSV file to load. Column number starts from 1. <wavegen_id> ::= WGEN1
:RECall:DBC[:STARt] [<file_name>] [, <serialbus>] (see page 679)	n/a	<file_name> ::= quoted ASCII string If extension included in file name, it must be ".dbc". <serialbus> ::= {SBUS<n>} <n> ::= 1 to (# of serial bus) in NR1 format
:RECall:FILename <base_name> (see page 680)	:RECall:FILename? (see page 680)	<base_name> ::= quoted ASCII string
:RECall:LDF[:STARt] [<file_name>] [, <serialbus>] (see page 681)	n/a	<file_name> ::= quoted ASCII string If extension included in file name, it must be ".ldf". <serialbus> ::= {SBUS<n>} <n> ::= 1 to (# of serial bus) in NR1 format
:RECall:MASK[:STARt] [{<internal_fname> | <external_fname>}] (see page 682)	n/a	<internal_fname> ::= quoted ASCII file name string beginning with "/User Files/" <external_fname> ::= quoted ASCII file name string beginning with "/USB1/" (or "/usb/"), "/USB2/", or "/USB3/" depending on the number of USB storage devices connected

Table 24 :RECall Commands Summary (continued)

Command	Query	Options and Query Returns
:RECall:PWD <path_name> (see page 683)	:RECall:PWD? (see page 683)	<path_name> ::= quoted ASCII string
:RECall:SETup[:STARt] [{<internal_fname> | <external_fname>}] (see page 684)	n/a	<internal_fname> ::= quoted ASCII file name string beginning with "/User Files/" <external_fname> ::= quoted ASCII file name string beginning with "/USB1/" (or "/usb/"), "/USB2/", or "/USB3/" depending on the number of USB storage devices connected
:RECall:WMEMory<r>[:STARt] [<file_name> | <data>] (see page 685)	n/a	<r> ::= 1 to (# ref waveforms) in NR1 format <file_name> ::= quoted ASCII string If extension included in file name, it must be ".h5". <data> ::= binary block data in IEEE 488.2 # format

Table 25 :SAVE Commands Summary

Command	Query	Options and Query Returns
:SAVE:ARBitrary:[STARt] [{<internal_fname> | <external_fname>}][, <wavegen_id>] (see page 691)	n/a	<internal_fname> ::= quoted ASCII file name string beginning with "/User Files/" <external_fname> ::= quoted ASCII file name string beginning with "/USB1/" (or "/usb/"), "/USB2/", or "/USB3/" depending on the number of USB storage devices connected <wavegen_id> ::= WGEN1
:SAVE:FILename <base_name> (see page 692)	:SAVE:FILename? (see page 692)	<base_name> ::= quoted ASCII string
:SAVE:IMAGe[:STARt] [<file_name>] (see page 693)	n/a	<file_name> ::= quoted ASCII string

Table 25 :SAVE Commands Summary (continued)

Command	Query	Options and Query Returns
:SAVE:IMAGe:FACTors {{0 | OFF} | {1 | ON}} (see page 694)	:SAVE:IMAGe:FACTors? (see page 694)	{0 | 1}
:SAVE:IMAGe:FORMat <format> (see page 695)	:SAVE:IMAGe:FORMat? (see page 695)	<format> ::= {{BMP | BMP24bit} | PNG | NONE}
:SAVE:LISTer[:STARt] [<file_name>] (see page 696)	n/a	<file_name> ::= quoted ASCII string
:SAVE:MASK[:STARt] [{<internal_fname> | <external_fname>}] (see page 697)	n/a	<internal_fname> ::= quoted ASCII file name string beginning with "/User Files/" <external_fname> ::= quoted ASCII file name string beginning with "/USB1/" (or "/usb/"), "/USB2/", or "/USB3/" depending on the number of USB storage devices connected
:SAVE:MULTi[:STARt] [<file_name>] (see page 698)	n/a	<file_name> ::= quoted ASCII string
:SAVE:PWD <path_name> (see page 699)	:SAVE:PWD? (see page 699)	<path_name> ::= quoted ASCII string
:SAVE:RESults:[STARt] [<file_spec>] (see page 700)	n/a	<file_name> ::= quoted ASCII string
:SAVE:RESults:FORMat: CURSor {{0 | OFF} | {1 | ON}} (see page 701)	:SAVE:RESults:FORMat: CURSor? (see page 701)	{0 | 1}
:SAVE:RESults:FORMat: HISTogram {{0 | OFF} | {1 | ON}} (see page 702)	:SAVE:RESults:FORMat: HISTogram? (see page 702)	{0 | 1}
:SAVE:RESults:FORMat: MASK {{0 | OFF} | {1 | ON}} (see page 703)	:SAVE:RESults:FORMat: MASK? (see page 703)	{0 | 1}
:SAVE:RESults:FORMat: MEASurement {{0 | OFF} | {1 | ON}} (see page 704)	:SAVE:RESults:FORMat: MEASurement? (see page 704)	{0 | 1}

Table 25 :SAVE Commands Summary (continued)

Command	Query	Options and Query Returns
:SAVE:RESults:FORMat:SEARch {{0 | OFF} | {1 | ON}} (see page 705)	:SAVE:RESults:FORMat:SEARch? (see page 705)	{0 | 1}
:SAVE:RESults:FORMat:SEGMented {{0 | OFF} | {1 | ON}} (see page 706)	:SAVE:RESults:FORMat:SEGMented? (see page 706)	{0 | 1}
:SAVE:SETup[:STARt] [{<internal_fname> | <external_fname>}] (see page 707)	n/a	<internal_fname> ::= quoted ASCII file name string beginning with "/User Files/" <external_fname> ::= quoted ASCII file name string beginning with "/USB1/" (or "/usb/"), "/USB2/", or "/USB3/" depending on the number of USB storage devices connected
:SAVE:WAVeform[:STARt] [<file_name>] (see page 708)	n/a	<file_name> ::= quoted ASCII string
:SAVE:WAVeform:FORMat <format> (see page 709)	:SAVE:WAVeform:FORMat? (see page 709)	<format> ::= {ASCiixy | CSV | BINary | NONE}
:SAVE:WAVeform:LENGth <length> (see page 710)	:SAVE:WAVeform:LENGth? (see page 710)	<length> ::= 100 to max. length; an integer in NR1 format
:SAVE:WAVeform:LENGth:MAX {{0 | OFF} | {1 | ON}} (see page 711)	:SAVE:WAVeform:LENGth:MAX? (see page 711)	{0 | 1}
:SAVE:WAVeform:SEGMented <option> (see page 712)	:SAVE:WAVeform:SEGMented? (see page 712)	<option> ::= {ALL | CURRent}

Table 25 :SAVE Commands Summary (continued)

Command	Query	Options and Query Returns
:SAVE:WMEMory:SOURce <source> (see page 713)	:SAVE:WMEMory:SOURce? (see page 713)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format NOTE: Only ADD or SUBtract math operations can be saved as reference waveforms. <return_value> ::= <source>
:SAVE:WMEMory[:STARt] [<file_name>] (see page 714)	n/a	<file_name> ::= quoted ASCII string If extension included in file name, it must be ".h5".

Table 26 General :SBUS<n> Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:DISPlay {{0 | OFF} | {1 | ON}} (see page 718)	:SBUS<n>:DISPlay? (see page 718)	{0 | 1}
:SBUS<n>:MODE <mode> (see page 719)	:SBUS<n>:MODE? (see page 719)	<mode> ::= {CAN | IIC | LIN | SPI | UART}

Table 27 :SBUS<n>:CAN Commands Summary

Command	Query	Options and Query Returns
n/a	:SBUS<n>:CAN:COUNt:ERRor? (see page 723)	<frame_count> ::= integer in NR1 format
n/a	:SBUS<n>:CAN:COUNt:OVERload? (see page 724)	<frame_count> ::= 0 in NR1 format
:SBUS<n>:CAN:COUNt:RESet (see page 725)	n/a	n/a
n/a	:SBUS<n>:CAN:COUNt:SPEC? (see page 726)	<spec_error_count> ::= integer in NR1 format

Table 27 :SBUS<n>:CAN Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:SBUS<n>:CAN:COUNt:TOTal? (see page 727)	<frame_count> ::= integer in NR1 format
n/a	:SBUS<n>:CAN:COUNt:UTILization? (see page 728)	<percent> ::= floating-point in NR3 format
:SBUS<n>:CAN:DISPlay <type> (see page 729)	:SBUS<n>:CAN:DISPlay? (see page 729)	<type> ::= {HEXadecimal | SYMBolic}
:SBUS<n>:CAN:FDSPoint <value> (see page 730)	:SBUS<n>:CAN:FDSPoint? (see page 730)	<value> ::= even numbered percentages from 30 to 90 in NR3 format.
:SBUS<n>:CAN:SAMPlepoint <percent> (see page 731)	:SBUS<n>:CAN:SAMPlepoint? (see page 731)	<percent> ::= 30.0 to 90.0 in NR3 format
:SBUS<n>:CAN:SIGNal:BAUDrate <baudrate> (see page 732)	:SBUS<n>:CAN:SIGNal:BAUDrate? (see page 732)	<baudrate> ::= integer from 10000 to 4000000 in 100 b/s increments, or 5000000
:SBUS<n>:CAN:SIGNal:DEFinition <value> (see page 733)	:SBUS<n>:CAN:SIGNal:DEFinition? (see page 733)	<value> ::= {CANH | CANL | RX | TX | DIFFerential | DIFL | DIFH}
:SBUS<n>:CAN:SIGNal:FDBaudrate <baudrate> (see page 734)	:SBUS<n>:CAN:SIGNal:FDBaudrate? (see page 734)	<baudrate> ::= integer from 10000 to 10000000 in 100 b/s increments.
:SBUS<n>:CAN:SIGNal:XLBaudrate <baudrate> (see page 735)	:SBUS<n>:CAN:SIGNal:XLBaudrate? (see page 735)	<baudrate> ::= integer from 10000 to 20000000 in 100 b/s increments.
:SBUS<n>:CAN:SOURce <source> (see page 736)	:SBUS<n>:CAN:SOURce? (see page 736)	<source> ::= {CHANnel<n> | DIGital<d> |} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:SBUS<n>:CAN:TRIGger <condition> (see page 737)	:SBUS<n>:CAN:TRIGger? (see page 738)	<condition> ::= {SOF | EOF | IDData | DATA | IDRemote | IDEither | ERRor | ACKerror | FORMerror | STUFferror | CRCerror | SPECerror | ALLerrors | BRSBit | CRCDbit | EBActive | EBPassive | OVERload | MESSage | MSIGnal}

Table 27 :SBUS<n>:CAN Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:CAN:TRIGger:IDFilter {{0 | OFF} | {1 | ON}} (see page 740)	:SBUS<n>:CAN:TRIGger:IDFilter? (see page 740)	{0 | 1}
:SBUS<n>:CAN:TRIGger:PATTern:DATA <string> (see page 741)	:SBUS<n>:CAN:TRIGger:PATTern:DATA? (see page 741)	<string> ::= "nn...n" where n ::= {0 | 1 | X | $} <string ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $}
:SBUS<n>:CAN:TRIGger:PATTern:DATA:DLC <dlc> (see page 742)	:SBUS<n>:CAN:TRIGger:PATTern:DATA:DLC? (see page 742)	<dlc> ::= integer between -1 (don't care) and 64, in NR1 format.
:SBUS<n>:CAN:TRIGger:PATTern:DATA:LENGth <length> (see page 743)	:SBUS<n>:CAN:TRIGger:PATTern:DATA:LENGth? (see page 743)	<length> ::= integer from 1 to 8 in NR1 format
:SBUS<n>:CAN:TRIGger:PATTern:DATA:STARt <start> (see page 744)	:SBUS<n>:CAN:TRIGger:PATTern:DATA:STARt? (see page 744)	<start> ::= integer between 0 and 63, in NR1 format.
:SBUS<n>:CAN:TRIGger:PATTern:ID <string> (see page 745)	:SBUS<n>:CAN:TRIGger:PATTern:ID? (see page 745)	<string> ::= "nn...n" where n ::= {0 | 1 | X | $} <string ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $}
:SBUS<n>:CAN:TRIGger:PATTern:ID:MODE <value> (see page 746)	:SBUS<n>:CAN:TRIGger:PATTern:ID:MODE? (see page 746)	<value> ::= {STANdard | EXTended}
:SBUS<n>:CAN:TRIGger:SYMBolic:MESSage <name> (see page 747)	:SBUS<n>:CAN:TRIGger:SYMBolic:MESSage? (see page 747)	<name> ::= quoted ASCII string
:SBUS<n>:CAN:TRIGger:SYMBolic:SIGNal <name> (see page 748)	:SBUS<n>:CAN:TRIGger:SYMBolic:SIGNal? (see page 748)	<name> ::= quoted ASCII string
:SBUS<n>:CAN:TRIGger:SYMBolic:VALue <data> (see page 749)	:SBUS<n>:CAN:TRIGger:SYMBolic:VALue? (see page 749)	<data> ::= value in NR3 format
:SBUS<n>:CAN:TRIGger:TYPE <type> (see page 750)	:SBUS<n>:CAN:TRIGger:TYPE? (see page 750)	<type> ::= {STANdard | FD | XL}

Table 27 :SBUS<n>:CAN Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:CAN:TYPE <type> (see page 751)	:SBUS<n>:CAN:TYPE? (see page 751)	<type> ::= {STANdard | FD | XFASt | XSIC}
:SBUS<n>:CAN:XLSPoint <value> (see page 752)	:SBUS<n>:CAN:XLSPoint? (see page 752)	<value> ::= even numbered percentages from 30 to 90 in NR3 format.

Table 28 :SBUS<n>:IIC Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:IIC:ASIZe <size> (see page 754)	:SBUS<n>:IIC:ASIZe? (see page 754)	<size> ::= {BIT7 | BIT8}
:SBUS<n>:IIC[:SOURce]:CLOCk <source> (see page 755)	:SBUS<n>:IIC[:SOURce]:CLOCk? (see page 755)	<source> ::= {CHANnel<n> | DIGital<d> } <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:SBUS<n>:IIC[:SOURce]:DATA <source> (see page 756)	:SBUS<n>:IIC[:SOURce]:DATA? (see page 756)	<source> ::= {CHANnel<n> | DIGital<d> } <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:SBUS<n>:IIC:TRIGger:PATTern:ADDRess <value> (see page 757)	:SBUS<n>:IIC:TRIGger:PATTern:ADDRess? (see page 757)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,..,9 | A,..,F}
:SBUS<n>:IIC:TRIGger:PATTern:DATA <value> (see page 758)	:SBUS<n>:IIC:TRIGger:PATTern:DATA? (see page 758)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,..,9 | A,..,F}
:SBUS<n>:IIC:TRIGger:PATTern:DATa2 <value> (see page 759)	:SBUS<n>:IIC:TRIGger:PATTern:DATa2? (see page 759)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,..,9 | A,..,F}

Table 28 :SBUS<n>:IIC Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:IIC:TRIGger:QUALifier <value> (see page 760)	:SBUS<n>:IIC:TRIGger:QUALifier? (see page 760)	<value> ::= {EQUal | NOTequal | LESSthan | GREaterthan}
:SBUS<n>:IIC:TRIGger[:TYPE] <type> (see page 761)	:SBUS<n>:IIC:TRIGger[:TYPE]? (see page 761)	<type> ::= {STARt | STOP | RESTart | ADDRess | ANACk | DNACk | NACKnowledge | REPRom | READ7 | WRITe7 | R7Data2 | W7Data2 | WRITe10}

Table 29 :SBUS<n>:LIN Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:LIN:DISPlay <type> (see page 765)	:SBUS<n>:LIN:DISPlay? (see page 765)	<type> ::= {HEXadecimal | SYMBolic}
:SBUS<n>:LIN:PARity {{0 | OFF} | {1 | ON}} (see page 766)	:SBUS<n>:LIN:PARity? (see page 766)	{0 | 1}
:SBUS<n>:LIN:SAMPlepoint <value> (see page 767)	:SBUS<n>:LIN:SAMPlepoint? (see page 767)	<value> ::= {60 | 62.5 | 68 | 70 | 75 | 80 | 87.5} in NR3 format
:SBUS<n>:LIN:SIGNal:BAUDrate <baudrate> (see page 768)	:SBUS<n>:LIN:SIGNal:BAUDrate? (see page 768)	<baudrate> ::= integer from 2400 to 625000 in 100 b/s increments
:SBUS<n>:LIN:SOURce <source> (see page 769)	:SBUS<n>:LIN:SOURce? (see page 769)	<source> ::= {CHANnel<n> | DIGital<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:SBUS<n>:LIN:STANdard <std> (see page 770)	:SBUS<n>:LIN:STANdard? (see page 770)	<std> ::= {LIN13 | LIN13NLC | LIN20}
:SBUS<n>:LIN:SYNCbreak <value> (see page 771)	:SBUS<n>:LIN:SYNCbreak? (see page 771)	<value> ::= integer = {11 | 12 | 13}
:SBUS<n>:LIN:TRIGger <condition> (see page 772)	:SBUS<n>:LIN:TRIGger? (see page 772)	<condition> ::= {SYNCbreak | ID | DATA}

Table 29 :SBUS<n>:LIN Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:LIN:TRIGger:ID <value> (see page 773)	:SBUS<n>:LIN:TRIGger:ID? (see page 773)	<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
:SBUS<n>:LIN:TRIGger:PATTern:DATA <string> (see page 774)	:SBUS<n>:LIN:TRIGger:PATTern:DATA? (see page 774)	<string> ::= "n" where n ::= 32-bit integer in unsigned decimal when <base> = DECimal <string> ::= "nn...n" where n ::= {0 | 1 | X | $} when <base> = BINary <string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $} when <base> = HEX
:SBUS<n>:LIN:TRIGger:PATTern:DATA:LENGth <length> (see page 776)	:SBUS<n>:LIN:TRIGger:PATTern:DATA:LENGth? (see page 776)	<length> ::= integer from 1 to 8 in NR1 format
:SBUS<n>:LIN:TRIGger:PATTern:FORMat <base> (see page 777)	:SBUS<n>:LIN:TRIGger:PATTern:FORMat? (see page 777)	<base> ::= {BINary | HEX | DECimal}
:SBUS<n>:LIN:TRIGger:SYMBolic:FRAMe <name> (see page 778)	:SBUS<n>:LIN:TRIGger:SYMBolic:FRAMe? (see page 778)	<name> ::= quoted ASCII string
:SBUS<n>:LIN:TRIGger:SYMBolic:SIGNal <name> (see page 779)	:SBUS<n>:LIN:TRIGger:SYMBolic:SIGNal? (see page 779)	<name> ::= quoted ASCII string
:SBUS<n>:LIN:TRIGger:SYMBolic:VALue <data> (see page 780)	:SBUS<n>:LIN:TRIGger:SYMBolic:VALue? (see page 780)	<data> ::= value in NR3 format

Table 30 :SBUS<n>:SPI Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:SPI:BITorder <order> (see page 783)	:SBUS<n>:SPI:BITorder? (see page 783)	<order> ::= {LSBFirst | MSBFirst}
:SBUS<n>:SPI:CLOCk:SLOPe <slope> (see page 784)	:SBUS<n>:SPI:CLOCk:SLOPe? (see page 784)	<slope> ::= {NEGative | POSitive}
:SBUS<n>:SPI:CLOCk:TIMeout <time_value> (see page 785)	:SBUS<n>:SPI:CLOCk:TIMeout? (see page 785)	<time_value> ::= time in seconds in NR3 format
:SBUS<n>:SPI:DELay <value> (see page 786)	:SBUS<n>:SPI:DELay? (see page 786)	<value> ::= {OFF | 2-63}
:SBUS<n>:SPI:FRAMing <value> (see page 787)	:SBUS<n>:SPI:FRAMing? (see page 787)	<value> ::= {CHIPselect | {NCHipselect | NOTC} | TIMeout}
:SBUS<n>:SPI:SOURce:CLOCk <source> (see page 788)	:SBUS<n>:SPI:SOURce:CLOCk? (see page 788)	<value> ::= {CHANnel<n> | DIGital<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:SBUS<n>:SPI:SOURce:FRAMe <source> (see page 789)	:SBUS<n>:SPI:SOURce:FRAMe? (see page 789)	<value> ::= {CHANnel<n> | DIGital<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:SBUS<n>:SPI:SOURce:MISO <source> (see page 790)	:SBUS<n>:SPI:SOURce:MISO? (see page 790)	<value> ::= {CHANnel<n> | DIGital<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:SBUS<n>:SPI:SOURce:MOSI <source> (see page 791)	:SBUS<n>:SPI:SOURce:MOSI? (see page 791)	<value> ::= {CHANnel<n> | DIGital<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format

Table 30 :SBUS<n>:SPI Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:SPI:TRIGger:PATTern:MISO:DATA <string> (see page 792)	:SBUS<n>:SPI:TRIGger:PATTern:MISO:DATA? (see page 792)	<string> ::= "nn...n" where n ::= {0 | 1 | X | $} <string ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $}
:SBUS<n>:SPI:TRIGger:PATTern:MISO:WIDTh <width> (see page 793)	:SBUS<n>:SPI:TRIGger:PATTern:MISO:WIDTh? (see page 793)	<width> ::= integer from 4 to 64 in NR1 format
:SBUS<n>:SPI:TRIGger:PATTern:MOSI:DATA <string> (see page 794)	:SBUS<n>:SPI:TRIGger:PATTern:MOSI:DATA? (see page 794)	<string> ::= "nn...n" where n ::= {0 | 1 | X | $} <string ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $}
:SBUS<n>:SPI:TRIGger:PATTern:MOSI:WIDTh <width> (see page 795)	:SBUS<n>:SPI:TRIGger:PATTern:MOSI:WIDTh? (see page 795)	<width> ::= integer from 4 to 64 in NR1 format
:SBUS<n>:SPI:TRIGger:TYPE <value> (see page 796)	:SBUS<n>:SPI:TRIGger:TYPE? (see page 796)	<value> ::= {MOSI | MISO}
:SBUS<n>:SPI:WIDTh <word_width> (see page 797)	:SBUS<n>:SPI:WIDTh? (see page 797)	<word_width> ::= integer 4-16 in NR1 format

Table 31 :SBUS<n>:UART Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:UART:BASE <base> (see page 801)	:SBUS<n>:UART:BASE? (see page 801)	<base> ::= {ASCii | BINary | HEX}
:SBUS<n>:UART:BAUDrate <baudrate> (see page 802)	:SBUS<n>:UART:BAUDrate? (see page 802)	<baudrate> ::= integer from 100 to 8000000, 10000000, or 12000000
:SBUS<n>:UART:BITorder <bitorder> (see page 803)	:SBUS<n>:UART:BITorder? (see page 803)	<bitorder> ::= {LSBFirst | MSBFirst}
n/a	:SBUS<n>:UART:COUNt:ERRor? (see page 804)	<frame_count> ::= integer in NR1 format
:SBUS<n>:UART:COUNt:RESet (see page 805)	n/a	n/a
n/a	:SBUS<n>:UART:COUNt:RXFRames? (see page 806)	<frame_count> ::= integer in NR1 format

Table 31 :SBUS<n>:UART Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:SBUS<n>:UART:COUNt:TXFRames? (see page 807)	<frame_count> ::= integer in NR1 format
:SBUS<n>:UART:FRAMing <value> (see page 808)	:SBUS<n>:UART:FRAMing? (see page 808)	<value> ::= {OFF | <decimal> | <nondecimal>} <decimal> ::= 8-bit integer from 0-255 (0x00-0xff) <nondecimal> ::= #Hnn where n ::= {0,..,9 | A,..,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary
:SBUS<n>:UART:PARity <parity> (see page 809)	:SBUS<n>:UART:PARity? (see page 809)	<parity> ::= {EVEN | ODD | NONE}
:SBUS<n>:UART:POLarity <polarity> (see page 810)	:SBUS<n>:UART:POLarity? (see page 810)	<polarity> ::= {HIGH | LOW}
:SBUS<n>:UART:SOURce:RX <source> (see page 811)	:SBUS<n>:UART:SOURce:RX? (see page 811)	<source> ::= {CHANnel<n> | DIGital<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:SBUS<n>:UART:SOURce:TX <source> (see page 812)	:SBUS<n>:UART:SOURce:TX? (see page 812)	<source> ::= {CHANnel<n> | DIGital<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:SBUS<n>:UART:TRIGger:BASE <base> (see page 813)	:SBUS<n>:UART:TRIGger:BASE? (see page 813)	<base> ::= {ASCii | HEX}
:SBUS<n>:UART:TRIGger:BURSt <value> (see page 814)	:SBUS<n>:UART:TRIGger:BURSt? (see page 814)	<value> ::= {OFF | 1 to 4096 in NR1 format}

Table 31 :SBUS<n>:UART Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:UART:TRIGger :DATA <value> (see page 815)	:SBUS<n>:UART:TRIGger :DATA? (see page 815)	<value> ::= 8-bit integer from 0-255 (0x00-0xff) in decimal, <hexadecimal>, <binary>, or <quoted_string> format <hexadecimal> ::= #Hnn where n ::= {0,..,9 | A,..,F} for hexadecimal <binary> ::= #Bnn...n where n ::= {0 | 1} for binary <quoted_string> ::= any of the 128 valid 7-bit ASCII characters (or standard abbreviations)
:SBUS<n>:UART:TRIGger :IDLE <time_value> (see page 816)	:SBUS<n>:UART:TRIGger :IDLE? (see page 816)	<time_value> ::= time from 1 us to 10 s in NR3 format
:SBUS<n>:UART:TRIGger :QUALifier <value> (see page 817)	:SBUS<n>:UART:TRIGger :QUALifier? (see page 817)	<value> ::= {EQUal | NOTequal | GREaterthan | LESSthan}
:SBUS<n>:UART:TRIGger :TYPE <value> (see page 818)	:SBUS<n>:UART:TRIGger :TYPE? (see page 818)	<value> ::= {RSTArt | RSTOp | RDATa | RD1 | RD0 | RDX | RXPArity | TXPArity | TSTArt | TSTOp | TDATa | TD1 | TD0 | TDX}
:SBUS<n>:UART:WIDTh <width> (see page 819)	:SBUS<n>:UART:WIDTh? (see page 819)	<width> ::= {5 | 6 | 7 | 8 | 9}

Table 32 General :SEARch Commands Summary

Command	Query	Options and Query Returns
n/a	:SEARch:COUNt? (see page 823)	<count> ::= an integer count value
:SEARch:EVENt <event_number> (see page 824)	:SEARch:EVENt? (see page 824)	<event_number> ::= the integer number of a found search event
:SEARch:MODE <value> (see page 825)	:SEARch:MODE? (see page 825)	<value> ::= {EDGE | GLITch | TRANsition | SERial{1 | 2} | PEAK}
:SEARch:STATe <value> (see page 826)	:SEARch:STATe? (see page 826)	<value> ::= {{0 | OFF} | {1 | ON}}

Table 33 :SEARch:EDGE Commands Summary

Command	Query	Options and Query Returns
:SEARch:EDGE:SLOPe <slope> (see page 828)	:SEARch:EDGE:SLOPe? (see page 828)	<slope> ::= {POSitive | NEGative | EITHer}
:SEARch:EDGE:SOURce <source> (see page 829)	:SEARch:EDGE:SOURce? (see page 829)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format

Table 34 :SEARch:GLITch Commands Summary

Command	Query	Options and Query Returns
:SEARch:GLITch:GREaterthan <greater_than_time>[suffix] (see page 831)	:SEARch:GLITch:GREaterthan? (see page 831)	<greater_than_time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:SEARch:GLITch:LESSthan <less_than_time>[suffix] (see page 832)	:SEARch:GLITch:LESSthan? (see page 832)	<less_than_time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:SEARch:GLITch:POLarity <polarity> (see page 833)	:SEARch:GLITch:POLarity? (see page 833)	<polarity> ::= {POSitive | NEGative}
:SEARch:GLITch:QUALifier <qualifier> (see page 834)	:SEARch:GLITch:QUALifier? (see page 834)	<qualifier> ::= {GREaterthan | LESSthan | RANGe}
:SEARch:GLITch:RANGe <less_than_time>[suffix], <greater_than_time>[suffix] (see page 835)	:SEARch:GLITch:RANGe? (see page 835)	<less_than_time> ::= 15 ns to 10 seconds in NR3 format <greater_than_time> ::= 10 ns to 9.99 seconds in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:SEARch:GLITch:SOURce <source> (see page 836)	:SEARch:GLITch:SOURce? (see page 836)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format

Table 35 :SEARch:PEAK Commands Summary

Command	Query	Options and Query Returns
:SEARch:PEAK:EXCursion <delta_level> (see page 838)	:SEARch:PEAK:EXCursion? (see page 838)	<delta_level> ::= required change in level to be recognized as a peak, in NR3 format.
:SEARch:PEAK:NPEaks <number> (see page 839)	:SEARch:PEAK:NPEaks? (see page 839)	<number> ::= max number of peaks to find, 1-11 in NR1 format.
:SEARch:PEAK:SOURce <source> (see page 840)	:SEARch:PEAK:SOURce? (see page 840)	<source> ::= {FUNCtion<m> | MATH<m>} (must be an FFT waveform) <m> ::= 1 to 4 in NR1 format
:SEARch:PEAK:THReshold <level> (see page 841)	:SEARch:PEAK:THReshold? (see page 841)	<level> ::= necessary level to be considered a peak, in NR3 format.

Table 36 :SEARch:TRANsition Commands Summary

Command	Query	Options and Query Returns
:SEARch:TRANsition:QUALifier <qualifier> (see page 843)	:SEARch:TRANsition:QUALifier? (see page 843)	<qualifier> ::= {GREaterthan | LESSthan}
:SEARch:TRANsition:SLOPe <slope> (see page 844)	:SEARch:TRANsition:SLOPe? (see page 844)	<slope> ::= {NEGative | POSitive}
:SEARch:TRANsition:SOURce <source> (see page 845)	:SEARch:TRANsition:SOURce? (see page 845)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:SEARch:TRANsition:TIME <time>[suffix] (see page 846)	:SEARch:TRANsition:TIME? (see page 846)	<time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}

Table 37 :SEARch:SERial:CAN Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:CAN:MODE <value> (see page 848)	:SEARch:SERial:CAN:MODE? (see page 848)	<value> ::= {IDEither | IDData | DATA | IDRemote | ERRor | ACKerror | FORMerror | STUFferror | CRCerror | ALLerrors | OVERload | MESSage | MSIGnal}
:SEARch:SERial:CAN:PATTern:DATA <string> (see page 850)	:SEARch:SERial:CAN:PATTern:DATA? (see page 850)	<string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X} for hexadecimal
:SEARch:SERial:CAN:PATTern:DATA:LENGth <length> (see page 851)	:SEARch:SERial:CAN:PATTern:DATA:LENGth? (see page 851)	<length> ::= integer from 1 to 8 in NR1 format
:SEARch:SERial:CAN:PATTern:ID <string> (see page 852)	:SEARch:SERial:CAN:PATTern:ID? (see page 852)	<string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X} for hexadecimal
:SEARch:SERial:CAN:PATTern:ID:MODE <value> (see page 853)	:SEARch:SERial:CAN:PATTern:ID:MODE? (see page 853)	<value> ::= {STANdard | EXTended}
:SEARch:SERial:CAN:SYMBolic:MESSage <name> (see page 854)	:SEARch:SERial:CAN:SYMBolic:MESSage? (see page 854)	<name> ::= quoted ASCII string
:SEARch:SERial:CAN:SYMBolic:SIGNal <name> (see page 855)	:SEARch:SERial:CAN:SYMBolic:SIGNal? (see page 855)	<name> ::= quoted ASCII string
:SEARch:SERial:CAN:SYMBolic:VALue <data> (see page 856)	:SEARch:SERial:CAN:SYMBolic:VALue? (see page 856)	<data> ::= value in NR3 format

Table 38 :SEARch:SERial:IIC Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:IIC:MODE <value> (see page 858)	:SEARch:SERial:IIC:MODE? (see page 858)	<value> ::= {RESTart | ADDRess | ANACk | NACKnowledge | REPRom | READ7 | WRITE7 | R7Data2 | W7Data2}
:SEARch:SERial:IIC:PATTern:ADDRess <value> (see page 860)	:SEARch:SERial:IIC:PATTern:ADDRess? (see page 860)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,..,9 | A,..,F}

Table 38 :SEARch:SERial:IIC Commands Summary (continued)

Command	Query	Options and Query Returns
:SEARch:SERial:IIC:PATTern:DATA <value> (see page 861)	:SEARch:SERial:IIC:PATTern:DATA? (see page 861)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,..,9 | A,..,F}
:SEARch:SERial:IIC:PATTern:DATA2 <value> (see page 862)	:SEARch:SERial:IIC:PATTern:DATA2? (see page 862)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,..,9 | A,..,F}
:SEARch:SERial:IIC:QUALifier <value> (see page 863)	:SEARch:SERial:IIC:QUALifier? (see page 863)	<value> ::= {EQUal | NOTequal | LESSthan | GREaterthan}

Table 39 :SEARch:SERial:LIN Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:LIN:ID <value> (see page 865)	:SEARch:SERial:LIN:ID? (see page 865)	<value> ::= 7-bit integer in decimal or <nondecimal> from 0-63 <nondecimal> ::= #Hnn where n ::= {0,..,9 | A,..,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary
:SEARch:SERial:LIN:MODE <value> (see page 866)	:SEARch:SERial:LIN:MODE? (see page 866)	<value> ::= {ID | DATA | ERRor}
:SEARch:SERial:LIN:PATTern:DATA <string> (see page 867)	:SEARch:SERial:LIN:PATTern:DATA? (see page 867)	When :SEARch:SERial:LIN:PATTern:FORMat DECimal, <string> ::= "n" where n ::= 32-bit integer in unsigned decimal, returns "$" if data has any don't cares When :SEARch:SERial:LIN:PATTern:FORMat HEX, <string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X}
:SEARch:SERial:LIN:PATTern:DATA:LENGth <length> (see page 868)	:SEARch:SERial:LIN:PATTern:DATA:LENGth? (see page 868)	<length> ::= integer from 1 to 8 in NR1 format
:SEARch:SERial:LIN:PATTern:FORMat <base> (see page 869)	:SEARch:SERial:LIN:PATTern:FORMat? (see page 869)	<base> ::= {HEX | DECimal}

Table 39 :SEARch:SERial:LIN Commands Summary (continued)

Command	Query	Options and Query Returns
:SEARch:SERial:LIN:SYMBolic:FRAMe <name> (see page 870)	:SEARch:SERial:LIN:SYMBolic:FRAMe? (see page 870)	<name> ::= quoted ASCII string
:SEARch:SERial:LIN:SYMBolic:SIGNal <name> (see page 871)	:SEARch:SERial:LIN:SYMBolic:SIGNal? (see page 871)	<name> ::= quoted ASCII string
:SEARch:SERial:LIN:SYMBolic:VALue <data> (see page 872)	:SEARch:SERial:LIN:SYMBolic:VALue? (see page 872)	<data> ::= value in NR3 format

Table 40 :SEARch:SERial:SPI Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:SPI:MODE <value> (see page 874)	:SEARch:SERial:SPI:MODE? (see page 874)	<value> ::= {MOSI | MISO}
:SEARch:SERial:SPI:PATTern:DATA <string> (see page 875)	:SEARch:SERial:SPI:PATTern:DATA? (see page 875)	<string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X}
:SEARch:SERial:SPI:PATTern:WIDTh <width> (see page 876)	:SEARch:SERial:SPI:PATTern:WIDTh? (see page 876)	<width> ::= integer from 1 to 10

Table 41 :SEARch:SERial:UART Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:UART:DATA <value> (see page 878)	:SEARch:SERial:UART:DATA? (see page 878)	<value> ::= 8-bit integer from 0-255 (0x00-0xff) in decimal, <hexadecimal>, or <binary> format <hexadecimal> ::= #Hnn where n ::= {0,..,9| A,..,F} for hexadecimal <binary> ::= #Bnn...n where n ::= {0 | 1} for binary

Table 41 :SEARch:SERial:UART Commands Summary (continued)

Command	Query	Options and Query Returns
:SEARch:SERial:UART:MODE <value> (see page 879)	:SEARch:SERial:UART:MODE? (see page 879)	<value> ::= {RDATa | RD1 | RD0 | RDX | TDATa | TD1 | TD0 | TDX | RXPArity | TXPArity | AERRor}
:SEARch:SERial:UART:QUALifier <value> (see page 881)	:SEARch:SERial:UART:QUALifier? (see page 881)	<value> ::= {EQUal | NOTequal | GREaterthan | LESSthan}

Table 42 General :STATus Commands Summary

Command	Query	Options and Query Returns
:STATus:PRESet (see page 887)	n/a	n/a

Table 43 :STATus:OPERation Commands Summary

Command	Query	Options and Query Returns
n/a	:STATus:OPERation:BIT <b>:CONDition? (see page 893)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:BIT <b>:ENABle <bit_value> (see page 894)	:STATus:OPERation:BIT <b>:ENABle? (see page 894)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:BIT <b>:NTRansition <bit_value> (see page 895)	:STATus:OPERation:BIT <b>:NTRansition? (see page 895)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:BIT <b>:PTRansition <bit_value> (see page 896)	:STATus:OPERation:BIT <b>:PTRansition? (see page 896)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation:BIT <b>[:EVENt]? (see page 897)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation:CONDition? (see page 898)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format

Table 43 :STATus:OPERation Commands Summary (continued)

Command	Query	Options and Query Returns
:STATus:OPERation:ENABle <sum_of_bits> (see page 899)	:STATus:OPERation:ENABle? (see page 899)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:NTRansition <sum_of_bits> (see page 900)	:STATus:OPERation:NTRansition? (see page 900)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:PTRansition <sum_of_bits> (see page 901)	:STATus:OPERation:PTRansition? (see page 901)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation[:EVENt]? (see page 902)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format

Table 44 :STATus:OPERation:ARM Commands Summary

Command	Query	Options and Query Returns
n/a	:STATus:OPERation:ARM:BIT<b>:CONDition? (see page 906)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:ARM:BIT<b>:ENABle <bit_value> (see page 907)	:STATus:OPERation:ARM:BIT<b>:ENABle? (see page 907)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:ARM:BIT<b>:NTRansition <bit_value> (see page 908)	:STATus:OPERation:ARM:BIT<b>:NTRansition? (see page 908)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:ARM:BIT<b>:PTRansition <bit_value> (see page 909)	:STATus:OPERation:ARM:BIT<b>:PTRansition? (see page 909)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation:ARM:BIT<b>[:EVENt]? (see page 910)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format

Table 44 :STATus:OPERation:ARM Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:STATus:OPERation:ARM :CONDition? (see page 911)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:ARM :ENABle <sum_of_bits> (see page 912)	:STATus:OPERation:ARM :ENABle? (see page 912)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:ARM :NTRansition <sum_of_bits> (see page 913)	:STATus:OPERation:ARM :NTRansition? (see page 913)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:ARM :PTRansition <sum_of_bits> (see page 914)	:STATus:OPERation:ARM :PTRansition? (see page 914)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation:ARM [:EVENt]? (see page 915)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format

Table 45 :STATus:OPERation:HARDware Commands Summary

Command	Query	Options and Query Returns
n/a	:STATus:OPERation:HARDware:BIT<b>:CONDition? (see page 919)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:HARDware:BIT<b>:ENABle <bit_value> (see page 920)	:STATus:OPERation:HARDware:BIT<b>:ENABle? (see page 920)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:HARDware:BIT<b>:NTRansition <bit_value> (see page 921)	:STATus:OPERation:HARDware:BIT<b>:NTRansition? (see page 921)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:HARDware:BIT<b>:PTRansition <bit_value> (see page 922)	:STATus:OPERation:HARDware:BIT<b>:PTRansition? (see page 922)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format

Table 45 :STATus:OPERation:HARDware Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:STATus:OPERation:HARDware:BIT<b>[:EVENt]? (see page 923)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation:HARDware:CONDition? (see page 924)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:HARDware:ENABle <sum_of_bits> (see page 925)	:STATus:OPERation:HARDware:ENABle? (see page 925)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:HARDware:NTRansition <sum_of_bits> (see page 926)	:STATus:OPERation:HARDware:NTRansition? (see page 926)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:HARDware:PTRansition <sum_of_bits> (see page 927)	:STATus:OPERation:HARDware:PTRansition? (see page 927)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation:HARDware[:EVENt]? (see page 928)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format

Table 46 :STATus:OPERation:LOCal Commands Summary

Command	Query	Options and Query Returns
n/a	:STATus:OPERation:LOCal:BIT<b>:CONDition? (see page 932)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:LOCal:BIT<b>:ENABle <bit_value> (see page 933)	:STATus:OPERation:LOCal:BIT<b>:ENABle? (see page 933)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:LOCal:BIT<b>:NTRansition <bit_value> (see page 934)	:STATus:OPERation:LOCal:BIT<b>:NTRansition? (see page 934)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format

Table 46 :STATus:OPERation:LOCal Commands Summary (continued)

Command	Query	Options and Query Returns
:STATus:OPERation:LOCal:BIT<b>:PTRansition <bit_value> (see page 935)	:STATus:OPERation:LOCal:BIT<b>:PTRansition? (see page 935)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation:LOCal:BIT<b>[:EVENt]? (see page 936)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation:LOCal:CONDition? (see page 937)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:LOCal:ENABle <sum_of_bits> (see page 938)	:STATus:OPERation:LOCal:ENABle? (see page 938)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:LOCal:NTRansition <sum_of_bits> (see page 939)	:STATus:OPERation:LOCal:NTRansition? (see page 939)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:LOCal:PTRansition <sum_of_bits> (see page 940)	:STATus:OPERation:LOCal:PTRansition? (see page 940)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation:LOCal[:EVENt]? (see page 941)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format

Table 47 :STATus:OPERation:MTESt Commands Summary

Command	Query	Options and Query Returns
n/a	:STATus:OPERation:MTESt:BIT<b>:CONDition? (see page 945)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:MTESt:BIT<b>:ENABle <bit_value> (see page 946)	:STATus:OPERation:MTESt:BIT<b>:ENABle? (see page 946)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format

Table 47 :STATus:OPERation:MTESt Commands Summary (continued)

Command	Query	Options and Query Returns
:STATus:OPERation:MTESt:BIT<b>:NTRansition <bit_value> (see page 947)	:STATus:OPERation:MTESt:BIT<b>:NTRansition? (see page 947)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:MTESt:BIT<b>:PTRansition <bit_value> (see page 948)	:STATus:OPERation:MTESt:BIT<b>:PTRansition? (see page 948)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation:MTESt:BIT<b>[:EVENt]? (see page 949)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation:MTESt:CONDition? (see page 950)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:MTESt:ENABle <sum_of_bits> (see page 951)	:STATus:OPERation:MTESt:ENABle? (see page 951)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:MTESt:NTRansition <sum_of_bits> (see page 952)	:STATus:OPERation:MTESt:NTRansition? (see page 952)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:MTESt:PTRansition <sum_of_bits> (see page 953)	:STATus:OPERation:MTESt:PTRansition? (see page 953)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation:MTESt[:EVENt]? (see page 954)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format

Table 48 :STATus:OPERation:OVERload Commands Summary

Command	Query	Options and Query Returns
n/a	:STATus:OPERation:OVERload:BIT<b>:CONDition? (see page 958)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:OVERload:BIT<b>:ENABle <bit_value> (see page 959)	:STATus:OPERation:OVERload:BIT<b>:ENABle? (see page 959)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:OVERload:BIT<b>:NTRansition <bit_value> (see page 960)	:STATus:OPERation:OVERload:BIT<b>:NTRansition? (see page 960)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:OVERload:BIT<b>:PTRansition <bit_value> (see page 961)	:STATus:OPERation:OVERload:BIT<b>:PTRansition? (see page 961)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation:OVERload:BIT<b>[:EVENt]? (see page 962)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation:OVERload:CONDition? (see page 963)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:OVERload:ENABle <sum_of_bits> (see page 964)	:STATus:OPERation:OVERload:ENABle? (see page 964)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:OVERload:NTRansition <sum_of_bits> (see page 965)	:STATus:OPERation:OVERload:NTRansition? (see page 965)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:OVERload:PTRansition <sum_of_bits> (see page 966)	:STATus:OPERation:OVERload:PTRansition? (see page 966)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation:OVERload[:EVENt]? (see page 967)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format

Table 49 :STATus:OPERation:OVERload:PFAult Commands Summary

Command	Query	Options and Query Returns
n/a	:STATus:OPERation:OVERload:PFAult:BIT<b>:CONDition? (see page 971)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:OVERload:PFAult:BIT<b>:ENABle <bit_value> (see page 972)	:STATus:OPERation:OVERload:PFAult:BIT<b>:ENABle? (see page 972)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:OVERload:PFAult:BIT<b>:NTRansition <bit_value> (see page 973)	:STATus:OPERation:OVERload:PFAult:BIT<b>:NTRansition? (see page 973)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:OVERload:PFAult:BIT<b>:PTRansition <bit_value> (see page 974)	:STATus:OPERation:OVERload:PFAult:BIT<b>:PTRansition? (see page 974)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation:OVERload:PFAult:BIT<b>[:EVENt]? (see page 975)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation:OVERload:PFAult:CONDition? (see page 976)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:OVERload:PFAult:ENABle <sum_of_bits> (see page 977)	:STATus:OPERation:OVERload:PFAult:ENABle? (see page 977)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:OVERload:PFAult:NTRansition <sum_of_bits> (see page 978)	:STATus:OPERation:OVERload:PFAult:NTRansition? (see page 978)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format

Table 49 :STATus:OPERation:OVERload:PFAult Commands Summary (continued)

Command	Query	Options and Query Returns
:STATus:OPERation:OVERload:PFAult:PTRansition <sum_of_bits> (see page 979)	:STATus:OPERation:OVERload:PFAult:PTRansition? (see page 979)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation:OVERload:PFAult[:EVENt]? (see page 980)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format

Table 50 :STATus:OPERation:POWer Commands Summary

Command	Query	Options and Query Returns
n/a	:STATus:OPERation:POWer:BIT<b>:CONDition? (see page 984)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:POWer:BIT<b>:ENABle <bit_value> (see page 985)	:STATus:OPERation:POWer:BIT<b>:ENABle? (see page 985)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:POWer:BIT<b>:NTRansition <bit_value> (see page 986)	:STATus:OPERation:POWer:BIT<b>:NTRansition? (see page 986)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:POWer:BIT<b>:PTRansition <bit_value> (see page 987)	:STATus:OPERation:POWer:BIT<b>:PTRansition? (see page 987)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation:POWer:BIT<b>[:EVENt]? (see page 988)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation:POWer:CONDition? (see page 989)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:POWer:ENABle <sum_of_bits> (see page 990)	:STATus:OPERation:POWer:ENABle? (see page 990)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format

Table 50 :STATus:OPERation:POWer Commands Summary (continued)

Command	Query	Options and Query Returns
:STATus:OPERation:POWer:NTRansition <sum_of_bits> (see page 991)	:STATus:OPERation:POWer:NTRansition? (see page 991)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:POWer:PTRansition <sum_of_bits> (see page 992)	:STATus:OPERation:POWer:PTRansition? (see page 992)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation:POWer[:EVENt]? (see page 993)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format

Table 51 :STATus:TRIGger Commands Summary

Command	Query	Options and Query Returns
n/a	:STATus:TRIGger:BIT<b>:CONDition? (see page 997)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:TRIGger:BIT<b>:ENABle <bit_value> (see page 998)	:STATus:TRIGger:BIT<b>:ENABle? (see page 998)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:TRIGger:BIT<b>:NTRansition <bit_value> (see page 999)	:STATus:TRIGger:BIT<b>:NTRansition? (see page 999)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:TRIGger:BIT<b>:PTRansition <bit_value> (see page 1000)	:STATus:TRIGger:BIT<b>:PTRansition? (see page 1000)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:TRIGger:BIT<b>[:EVENt]? (see page 1001)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:TRIGger:CONDition? (see page 1002)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format

Table 51 :STATus:TRIGger Commands Summary (continued)

Command	Query	Options and Query Returns
:STATus:TRIGger:ENABle <sum_of_bits> (see page 1003)	:STATus:TRIGger:ENABle? (see page 1003)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:TRIGger:NTRansition <sum_of_bits> (see page 1004)	:STATus:TRIGger:NTRansition? (see page 1004)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:TRIGger:PTRansition <sum_of_bits> (see page 1005)	:STATus:TRIGger:PTRansition? (see page 1005)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:TRIGger[:EVENt]? (see page 1006)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format

Table 52 :STATus:USER Commands Summary

Command	Query	Options and Query Returns
n/a	:STATus:USER:BIT<b>:CONDition? (see page 1010)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:USER:BIT<b>:ENABle <bit_value> (see page 1011)	:STATus:USER:BIT<b>:ENABle? (see page 1011)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:USER:BIT<b>:NTRansition <bit_value> (see page 1012)	:STATus:USER:BIT<b>:NTRansition? (see page 1012)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:USER:BIT<b>:PTRansition <bit_value> (see page 1013)	:STATus:USER:BIT<b>:PTRansition? (see page 1013)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:USER:BIT<b>[:EVENt]? (see page 1014)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format

Table 52 :STATus:USER Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:STATus:USER:CONDition? (see page 1015)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:USER:ENABle <sum_of_bits> (see page 1016)	:STATus:USER:ENABle? (see page 1016)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:USER:NTRansition <sum_of_bits> (see page 1017)	:STATus:USER:NTRansition? (see page 1017)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:USER:PTRansition <sum_of_bits> (see page 1018)	:STATus:USER:PTRansition? (see page 1018)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:USER[:EVENt]? (see page 1019)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format

Table 53 :SYSTem Commands Summary

Command	Query	Options and Query Returns
:SYSTem:DATE <date> (see page 1025)	:SYSTem:DATE? (see page 1025)	<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}
n/a	:SYSTem:DIDentifier? (see page 1026)	n/a
:SYSTem:DSP <string> (see page 1027)	n/a	<string> ::= up to 75 characters as a quoted ASCII string

Table 53 :SYSTem Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:SYSTem:ERRor:ALL? (see page 1028)	<errors_list> ::= comma-separated list of all errors from the queue (codes and strings)
n/a	:SYSTem:ERRor:CODE[:NEXT]? (see page 1029)	<error_code> ::= the next error (code only) from the queue
n/a	:SYSTem:ERRor:COUNt? (see page 1030)	<count> ::= an integer number of errors in the queue
n/a	:SYSTem:ERRor:EXTended? <type> (see page 1031)	<type> ::= {INFO | WARNING | ERROR | FATAL | MESSAGE | ALL} <errors_list> ::= returns a comma-separated list of all errors (strings, no codes) of the specified type from the queue
n/a	:SYSTem:ERRor[:NEXT]? (see page 1032)	<error_number>,<error_string> ::= the next error (code and string) from the queue <error_number> ::= an integer error code <error string> ::= quoted ASCII string. See Error Messages (see page 1245).
n/a	:SYSTem:HELP:HEADers? (see page 1034)	<binary_block> ::= newline-separated list of supported SCPI commands/queries
n/a	:SYSTem:HID? (see page 1035)	n/a
:SYSTem:LOCK <value> (see page 1036)	:SYSTem:LOCK? (see page 1036)	<value> ::= {{1 | ON} | {0 | OFF}}
n/a	:SYSTem:LOCK:OWNer? (see page 1037)	<session_string> ::= quoted session string or "NONE"
:SYSTem:LOCK:RELease (see page 1038)	n/a	n/a
n/a	:SYSTem:LOCK:REQuest? (see page 1039)	{1 | 0} (for granted or not)
:SYSTem:PERSona[:MANufacturer] <manufacturer_string> (see page 1040)	:SYSTem:PERSona[:MANufacturer]? (see page 1040)	<manufacturer_string> ::= quoted ASCII string, up to 63 characters

Table 53 :SYSTem Commands Summary (continued)

Command	Query	Options and Query Returns
:SYSTem:PERSona[:MANufacturer]:DEFault (see page 1041)	n/a	Sets manufacturer string to "KEYSIGHT TECHNOLOGIES"
:SYSTem:POWercycle (see page 1042)	n/a	n/a
:SYSTem:PRECision:LENGth <length> (see page 1043)	:SYSTem:PRECision:LENGth? (see page 1043)	<length> ::= {64K | 128K | 256K | 512K | 1M | 1.5M | 2M | 4M | 8M | 12M | 16M | 24M | 32M}
:SYSTem:PRESet (see page 1044)	n/a	See :SYSTem:PRESet (see page 1044)
:SYSTem:PROTection:LOCK <value> (see page 1047)	:SYSTem:PROTection:LOCK? (see page 1047)	<value> ::= {{1 | ON} | {0 | OFF}}
:SYSTem:RLOGger <setting>[,<file_name>[,<write_mode>]] (see page 1048)	n/a	<setting> ::= {{0 | OFF} | {1 | ON}} <file_name> ::= quoted ASCII string <write_mode> ::= {CREate | APPend}
:SYSTem:RLOGger:DESTination <dest> (see page 1049)	:SYSTem:RLOGger:DESTination? (see page 1049)	<dest> ::= {FILE | SCReen | BOTH}
:SYSTem:RLOGger:DISPlay {{0 | OFF} | {1 | ON}} (see page 1050)	:SYSTem:RLOGger:DISPlay? (see page 1050)	<setting> ::= {0 | 1}
:SYSTem:RLOGger:FNAMe <file_name> (see page 1051)	:SYSTem:RLOGger:FNAMe? (see page 1051)	<file_name> ::= quoted ASCII string
:SYSTem:RLOGger:STATe {{0 | OFF} | {1 | ON}} (see page 1052)	:SYSTem:RLOGger:STATe? (see page 1052)	<setting> ::= {0 | 1}
:SYSTem:RLOGger:TRANsparent {{0 | OFF} | {1 | ON}} (see page 1053)	:SYSTem:RLOGger:TRANsparent? (see page 1053)	<setting> ::= {0 | 1}
:SYSTem:RLOGger:WMODe <write_mode> (see page 1054)	:SYSTem:RLOGger:WMODe? (see page 1054)	<write_mode> ::= {CREate | APPend}

Table 53 :SYSTem Commands Summary (continued)

Command	Query	Options and Query Returns
:SYSTem:SETup <setup_data> (see page 1055)	:SYSTem:SETup? (see page 1055)	<setup_data> ::= data in IEEE 488.2 # format.
:SYSTem:SHUTdown (see page 1057)	n/a	n/a
:SYSTem:TIME <time> (see page 1058)	:SYSTem:TIME? (see page 1058)	<time> ::= hours,minutes,seconds in NR1 format
:SYSTem:TIME:DSAVing <value> (see page 1059)	:SYSTem:TIME:DSAVing? (see page 1059)	<value> ::= {{1 | ON} | {0 | OFF}}
:SYSTem:TIME:NTPRotocol {{0 | OFF} | {1 | ON}} (see page 1060)	:SYSTem:TIME:NTPRotocol? (see page 1060)	<setting> ::= {0 | 1}
:SYSTem:TIME:ZONE <UTC_P_or_M_HHMM_offset> (see page 1061)	:SYSTem:TIME:ZONE? (see page 1061)	<UTC_P_or_M_HHMM_offset> ::= {UP0000 | UP0100 | UP0200 | UP0300 | UP0330 | UP0400 | UP0430 | UP0500 | UP0530 | UP0600 | UP0700 | UP0800 | UP0845 | UP0900 | UP0930 | UP1000 | UP1030 | UP1100 | UP1200 | UM0100 | UM0200 | UM0300 | UM0330 | UM0400 | UM0500 | UM0600 | UM0700 | UM0800 | UM0900 | UM1000 | UM1100 | UM1200}
:SYSTem:TOUCh {{1 | ON} | {0 | OFF}} (see page 1062)	:SYSTem:TOUCh? (see page 1062)	{1 | 0}
n/a	:SYSTem:VERSion? (see page 1063)	<SCPI_version_implemented> ::= 1999.0

Table 54 :TIMebase Commands Summary

Command	Query	Options and Query Returns
:TIMebase[:MAIN]:POSition <pos> (see page 1067)	:TIMebase[:MAIN]:POSition? (see page 1067)	<pos> ::= time from the trigger event to the display reference point in NR3 format
:TIMebase[:MAIN]:RANGe <range_value> (see page 1068)	:TIMebase[:MAIN]:RANGe? (see page 1068)	<range_value> ::= time for 10 div in seconds in NR3 format

Table 54 :TIMebase Commands Summary (continued)

Command	Query	Options and Query Returns
:TIMebase[:MAIN]:SCALe <scale_value> (see page 1069)	:TIMebase[:MAIN]:SCALe? (see page 1069)	<scale_value> ::= time/div in seconds in NR3 format
:TIMebase:MODE <value> (see page 1070)	:TIMebase:MODE? (see page 1070)	<value> ::= {MAIN | WINDow}
:TIMebase:REFerence {LEFT | CENTer | RIGHt | CUSTom} (see page 1071)	:TIMebase:REFerence? (see page 1071)	<return_value> ::= {LEFT | CENTer | RIGHt | CUSTom}
:TIMebase:REFerence:LOCation <loc> (see page 1072)	:TIMebase:REFerence:LOCation? (see page 1072)	<loc> ::= 0.0 to 1.0 in NR3 format
:TIMebase:VERNier {{0 | OFF} | {1 | ON}} (see page 1073)	:TIMebase:VERNier? (see page 1073)	{0 | 1}
:TIMebase:WINDow:POSition <pos> (see page 1074)	:TIMebase:WINDow:POSition? (see page 1074)	<pos> ::= time from the trigger event to the zoomed view reference point in NR3 format
:TIMebase:WINDow:RANGe <range_value> (see page 1075)	:TIMebase:WINDow:RANGe? (see page 1075)	<range value> ::= range value in seconds in NR3 format for the zoomed window
:TIMebase:WINDow:SCALe <scale_value> (see page 1076)	:TIMebase:WINDow:SCALe? (see page 1076)	<scale_value> ::= scale value in seconds in NR3 format for the zoomed window

Table 55 General :TRIGger Commands Summary

Command	Query	Options and Query Returns
:TRIGger:FORCe (see page 1081)	n/a	n/a
:TRIGger:HFReject {{0 | OFF} | {1 | ON}} (see page 1082)	:TRIGger:HFReject? (see page 1082)	{0 | 1}
:TRIGger:HOLDoff <holdoff_time> (see page 1083)	:TRIGger:HOLDoff? (see page 1083)	<holdoff_time> ::= 60 ns to 10 s in NR3 format
:TRIGger:HOLDoff:MAXimum <max_holdoff> (see page 1084)	:TRIGger:HOLDoff:MAXimum? (see page 1084)	<max_holdoff> ::= maximum holdoff time in seconds in NR3 format

Table 55 General :TRIGger Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:HOLDoff:MINimum <min_holdoff> (see page 1085)	:TRIGger:HOLDoff:MINimum? (see page 1085)	<min_holdoff> ::= minimum holdoff time in seconds in NR3 format
:TRIGger:HOLDoff:RANDom {{0 | OFF} | {1 | ON}} (see page 1086)	:TRIGger:HOLDoff:RANDom? (see page 1086)	<setting> ::= {0 | 1}
:TRIGger:JFRee {{0 | OFF} | {1 | ON}} (see page 1087)	:TRIGger:JFRee? (see page 1087)	{0 | 1}
:TRIGger:LEVel:ASETup (see page 1088)	n/a	n/a
:TRIGger:LEVel:HIGH <level>, <source> (see page 1089)	:TRIGger:LEVel:HIGH? <source> (see page 1089)	<level> ::= .75 x full-scale voltage from center screen in NR3 format. <source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:LEVel:LOW <level>, <source> (see page 1090)	:TRIGger:LEVel:LOW? <source> (see page 1090)	<level> ::= .75 x full-scale voltage from center screen in NR3 format. <source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:MODE <mode> (see page 1091)	:TRIGger:MODE? (see page 1091)	<mode> ::= {EDGE | GLITch | PATTern | DELay | OR | RUNT | SHOLd | TRANsition | SBUS{1 | 2}} <return_value> ::= {<mode>}
:TRIGger:NREJect {{0 | OFF} | {1 | ON}} (see page 1092)	:TRIGger:NREJect? (see page 1092)	{0 | 1}
:TRIGger:SWEep <sweep> (see page 1093)	:TRIGger:SWEep? (see page 1093)	<sweep> ::= {AUTO | NORMal}

Table 56 :TRIGger[:EDGE] Commands Summary

Command	Query	Options and Query Returns
:TRIGger[:EDGE]:COUPling {AC | DC | LFReject} (see page 1095)	:TRIGger[:EDGE]:COUPling? (see page 1095)	{AC | DC | LFReject}
:TRIGger[:EDGE]:LEVel <level> [,<source>] (see page 1096)	:TRIGger[:EDGE]:LEVel? [<source>] (see page 1096)	For internal triggers, <level> ::= .75 x full-scale voltage from center screen in NR3 format. For external triggers, <level> ::= ±(external range setting) in NR3 format. For digital channels, <level> ::= ±8 V. <source> ::= {CHANnel<n> | DIGital<d> | EXTernal } <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:TRIGger[:EDGE]:REJect {OFF | LFReject | HFReject} (see page 1097)	:TRIGger[:EDGE]:REJect? (see page 1097)	{OFF | LFReject | HFReject}
:TRIGger[:EDGE]:SLOPe <polarity> (see page 1098)	:TRIGger[:EDGE]:SLOPe? (see page 1098)	<polarity> ::= {POSitive | NEGative | EITHer | ALTernate}
:TRIGger[:EDGE]:SOURce <source> (see page 1099)	:TRIGger[:EDGE]:SOURce? (see page 1099)	<source> ::= {CHANnel<n> | DIGital<d> | EXTernal | LINE | WGEN} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format

Table 57 :TRIGger:GLITch Commands Summary

Command	Query	Options and Query Returns
:TRIGger:GLITch:GREaterthan <greater_than_time>[suffix] (see page 1102)	:TRIGger:GLITch:GREaterthan? (see page 1102)	<greater_than_time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:GLITch:LESSthan <less_than_time>[suffix] (see page 1103)	:TRIGger:GLITch:LESSthan? (see page 1103)	<less_than_time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:GLITch:LEVel <level> [<source>] (see page 1104)	:TRIGger:GLITch:LEVel? (see page 1104)	For internal triggers, <level> ::= .75 x full-scale voltage from center screen in NR3 format. For digital channels, <level> ::= ±8 V. <source> ::= {CHANnel<n> | DIGital<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:TRIGger:GLITch:POLarity <polarity> (see page 1105)	:TRIGger:GLITch:POLarity? (see page 1105)	<polarity> ::= {POSitive | NEGative}
:TRIGger:GLITch:QUALifier <qualifier> (see page 1106)	:TRIGger:GLITch:QUALifier? (see page 1106)	<qualifier> ::= {GREaterthan | LESSthan | RANGe}
:TRIGger:GLITch:RANGe <less_than_time>[suffix], <greater_than_time>[suffix] (see page 1107)	:TRIGger:GLITch:RANGe? (see page 1107)	<less_than_time> ::= 15 ns to 10 seconds in NR3 format <greater_than_time> ::= 10 ns to 9.99 seconds in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:GLITch:SOURce <source> (see page 1108)	:TRIGger:GLITch:SOURce? (see page 1108)	<source> ::= {CHANnel<n> | DIGital<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format

Table 58 :TRIGger:OR Commands Summary

Command	Query	Options and Query Returns
:TRIGger:OR <string> (see page 1110)	:TRIGger:OR? (see page 1110)	<string> ::= "nn...n" where n ::= {R | F | E | X} R = rising edge, F = falling edge, E = either edge, X = don't care. Each character in the string is for an analog or digital channel as shown on the front panel display.

Table 59 :TRIGger:PATTern Commands Summary

Command	Query	Options and Query Returns
:TRIGger:PATTern <string>[,<edge_source>,<edge>] (see page 1112)	:TRIGger:PATTern? (see page 1113)	<string> ::= "nn...n" where n ::= {0 | 1 | X | R | F} when <base> = ASCii <string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $} when <base> = HEX <edge_source> ::= {CHANnel<n> | DIGital<d> | NONE} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <edge> ::= {POSitive | NEGative}
:TRIGger:PATTern:FORMat <base> (see page 1114)	:TRIGger:PATTern:FORMat? (see page 1114)	<base> ::= {ASCii | HEX}
:TRIGger:PATTern:GREaterthan <greater_than_time>[suffix] (see page 1115)	:TRIGger:PATTern:GREaterthan? (see page 1115)	<greater_than_time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:PATTern:LESSthan <less_than_time>[suffix] (see page 1116)	:TRIGger:PATTern:LESSthan? (see page 1116)	<less_than_time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}

Table 59 :TRIGger:PATTern Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:PATTern:QUALifier <qualifier> (see page 1117)	:TRIGger:PATTern:QUALifier? (see page 1117)	<qualifier> ::= {ENTered | GREaterthan | LESSthan | INRange | OUTRange | TIMeout}
:TRIGger:PATTern:RANGe <less_than_time>[suffix], <greater_than_time>[suffix] (see page 1118)	:TRIGger:PATTern:RANGe? (see page 1118)	<less_than_time> ::= 15 ns to 10 seconds in NR3 format <greater_than_time> ::= 10 ns to 9.99 seconds in NR3 format [suffix] ::= {s | ms | us | ns | ps}

Table 60 :TRIGger:RUNT Commands Summary

Command	Query	Options and Query Returns
:TRIGger:RUNT:POLarity <polarity> (see page 1120)	:TRIGger:RUNT:POLarity? (see page 1120)	<polarity> ::= {POSitive | NEGative}
:TRIGger:RUNT:QUALifier <qualifier> (see page 1121)	:TRIGger:RUNT:QUALifier? (see page 1121)	<qualifier> ::= {GREaterthan | LESSthan | NONE}
:TRIGger:RUNT:SOURce <source> (see page 1122)	:TRIGger:RUNT:SOURce? (see page 1122)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:RUNT:TIME <time>[suffix] (see page 1123)	:TRIGger:RUNT:TIME? (see page 1123)	<time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}

Table 61 :TRIGger:SHOLd Commands Summary

Command	Query	Options and Query Returns
:TRIGger:SHOLd:SLOPe <slope> (see page 1125)	:TRIGger:SHOLd:SLOPe? (see page 1125)	<slope> ::= {NEGative | POSitive}
:TRIGger:SHOLd:SOURce:CLOCk <source> (see page 1126)	:TRIGger:SHOLd:SOURce:CLOCk? (see page 1126)	<source> ::= {CHANnel<n> | DIGital<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format

Table 61 :TRIGger:SHOLd Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:SHOLd:SOURce :DATA <source> (see page 1127)	:TRIGger:SHOLd:SOURce :DATA? (see page 1127)	<source> ::= {CHANnel<n> | DIGital<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:TRIGger:SHOLd:TIME:HOLD <time>[suffix] (see page 1128)	:TRIGger:SHOLd:TIME:HOLD? (see page 1128)	<time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:SHOLd:TIME:SETup <time>[suffix] (see page 1129)	:TRIGger:SHOLd:TIME:SETup? (see page 1129)	<time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}

Table 62 :TRIGger:TRANsition Commands Summary

Command	Query	Options and Query Returns
:TRIGger:TRANsition:QUALifier <qualifier> (see page 1131)	:TRIGger:TRANsition:QUALifier? (see page 1131)	<qualifier> ::= {GREaterthan | LESSthan}
:TRIGger:TRANsition:SLOPe <slope> (see page 1132)	:TRIGger:TRANsition:SLOPe? (see page 1132)	<slope> ::= {NEGative | POSitive}
:TRIGger:TRANsition:SOURce <source> (see page 1133)	:TRIGger:TRANsition:SOURce? (see page 1133)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:TRANsition:TIME <time>[suffix] (see page 1134)	:TRIGger:TRANsition:TIME? (see page 1134)	<time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}

Table 63 :TRIGger:ZONE Commands Summary

Command	Query	Options and Query Returns
:TRIGger:ZONE<z>:LOGic {AND | OR} (see page 1136)	:TRIGger:ZONE<z>:LOGic? (see page 1140)	{AND | OR} <z> ::= 2-4 in NR1 format
:TRIGger:ZONE<z>:MODE <mode> (see page 1137)	:TRIGger:ZONE<z>:MODE? (see page 1137)	<mode> ::= {INTersect | NOTintersect} <z> ::= 1-4 in NR1 format
:TRIGger:ZONE<z>:PLACement <width>, <height>, <x_center>, <y_center> (see page 1138)	:TRIGger:ZONE<z>:PLACement? (see page 1138)	<width> ::= width of zone in seconds <height> ::= height of zone in volts <x_center> ::= center of zone in seconds <y_center> ::= center of zone in volts <z> ::= 1-4 in NR1 format
:TRIGger:ZONE<z>:SOURce <source> (see page 1139)	:TRIGger:ZONE<z>:SOURce? (see page 1139)	<source> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format <z> ::= 1-4 in NR1 format
:TRIGger:ZONE<z>:STATe {{0 | OFF} | {1 | ON}} (see page 1140)	:TRIGger:ZONE<z>:STATe? (see page 1140)	{0 | 1} <z> ::= 1-4 in NR1 format
n/a	:TRIGger:ZONE<z>:VALidity? (see page 1141)	<value> ::= {VALid | INValid | OSCReen} <z> ::= 1-4 in NR1 format

Table 64 :WAVeform Commands Summary

Command	Query	Options and Query Returns
:WAVeform:BYTeorder <value> (see page 1151)	:WAVeform:BYTeorder? (see page 1151)	<value> ::= {LSBFirst | MSBFirst}
n/a	:WAVeform:COUNt? (see page 1152)	<count> ::= an integer from 1 to 65536 in NR1 format

Table 64 :WAVeform Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:WAVeform:DATA? (see page 1153)	<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> (see page 1155)	:WAVeform:FORMat? (see page 1155)	<value> ::= {WORD | BYTE | ASCII}
:WAVeform:POINts <wfm_points> (see page 1156)	:WAVeform:POINts? (see page 1156)	<wfm_points> ::= <#_points> <#_points> ::= an integer in NR1 format
:WAVeform:POINts:MODE <points_mode> (see page 1158)	:WAVeform:POINts:MODE? (see page 1159)	<points_mode> ::= {NORMal | MAXimum | RAW}
n/a	:WAVeform:PREamble? (see page 1160)	<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 • 3 for AVERage type <count> ::= Average count, or 1 if PEAK detect type or NORMal; an integer in NR1 format

Table 64 :WAVeform Commands Summary (continued)

Command	Query	Options and Query Returns
:WAVeform:SEGMented:ALL {{0 | OFF} | {1 | ON}} (see page 1163)	:WAVeform:SEGMented:ALL? (see page 1163)	<setting> ::= {0 | 1}
n/a	:WAVeform:SEGMented:COUNt? (see page 1164)	<count> ::= an integer from 2 to 1000 in NR1 format
n/a	:WAVeform:SEGMented:TTAG? (see page 1165)	<time_tag> ::= in NR3 format
n/a	:WAVeform:SEGMented:XLISt? <xlist_type> (see page 1166)	<xlist_type> ::= {RELXorigin | ABSXorigin | TTAG} <return_value> ::= X-info for all segments
:WAVeform:SOURce <source> (see page 1167)	:WAVeform:SOURce? (see page 1167)	<source> ::= {CHANnel<n> | POD{1 | 2} | BUS{1 | 2} | FUNCtion<m> | MATH<m> | FFT | SBUS} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format
:WAVeform:SOURce:SUBSource <subsource> (see page 1171)	:WAVeform:SOURce:SUBSource? (see page 1171)	<subsource> ::= {{SUB0 | RX | MOSI} | {SUB1 | TX | MISO}}
n/a	:WAVeform:TYPE? (see page 1172)	<return_mode> ::= {NORM | PEAK | AVER}
:WAVeform:UNSigned {{0 | OFF} | {1 | ON}} (see page 1173)	:WAVeform:UNSigned? (see page 1173)	{0 | 1}
:WAVeform:VIEW <view> (see page 1174)	:WAVeform:VIEW? (see page 1174)	<view> ::= {MAIN | ALL}
n/a	:WAVeform:XINCrement? (see page 1175)	<return_value> ::= x-increment in the current preamble in NR3 format
n/a	:WAVeform:XORigin? (see page 1176)	<return_value> ::= x-origin value in the current preamble in NR3 format
n/a	:WAVeform:XREFerence? (see page 1177)	<return_value> ::= 0 (x-reference value in the current preamble in NR1 format)
n/a	:WAVeform:YINCrement? (see page 1178)	<return_value> ::= y-increment value in the current preamble in NR3 format

Table 64 :WAVeform Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:WAVeform:YORigin? (see page 1179)	<return_value> ::= y-origin in the current preamble in NR3 format
n/a	:WAVeform:YREFerence? (see page 1180)	<return_value> ::= y-reference value in the current preamble in NR1 format

Table 65 :WGEN<w> Commands Summary

Command	Query	Options and Query Returns
:WGEN<w>:ARBitrary:BYTeorder <order> (see page 1186)	:WGEN<w>:ARBitrary:BYTeorder? (see page 1186)	<order> ::= {MSBFirst | LSBFirst} <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:ARBitrary:DATA {<binary> | <value>, <value> ...} (see page 1187)	n/a	<binary> ::= floating point values between -1.0 to +1.0 in IEEE 488.2 binary block format <value> ::= floating point values between -1.0 to +1.0 in comma-separated format <w> ::= 1 to (# WaveGen outputs) in NR1 format
n/a	:WGEN<w>:ARBitrary:DATA:ATTRibute:POINts? (see page 1190)	<points> ::= number of points in NR1 format <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:ARBitrary:DATA:CLEar (see page 1191)	n/a	<w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:ARBitrary:DATA:DAC {<binary> | <value>, <value> ...} (see page 1192)	n/a	<binary> ::= decimal 16-bit integer values between -512 to +511 in IEEE 488.2 binary block format <value> ::= decimal integer values between -512 to +511 in comma-separated NR1 format <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:ARBitrary:INTerpolate {{0 | OFF} | {1 | ON}} (see page 1193)	:WGEN<w>:ARBitrary:INTerpolate? (see page 1193)	{0 | 1} <w> ::= 1 to (# WaveGen outputs) in NR1 format

Table 65 :WGEN<w> Commands Summary (continued)

Command	Query	Options and Query Returns
:WGEN<w>:ARBitrary:STORe <source> (see page 1194)	n/a	<source> ::= {CHANnel<n> | WMEMory<r> | FUNCtion<m> | FFT | MATH<m>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:DCMode <mode> (see page 1195)	:WGEN<w>:DCMode? (see page 1195)	<mode> ::= {PRECise | WIDerange}
:WGEN<w>:FREQuency <frequency> (see page 1196)	:WGEN<w>:FREQuency? (see page 1196)	<frequency> ::= frequency in Hz in NR3 format <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:FUNCtion <signal> (see page 1197)	:WGEN<w>:FUNCtion? (see page 1201)	<signal> ::= {SINusoid | SQUare | RAMP | PULSe | NOISe | DC | SINC | EXPRise | EXPFall | CARDiac | GAUSsian | ARBitrary} <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:FUNCtion:PULSe:WIDTh <width> (see page 1202)	:WGEN<w>:FUNCtion:PULSe:WIDTh? (see page 1202)	<width> ::= pulse width in seconds in NR3 format <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:FUNCtion:SQUare:DCYCle <percent> (see page 1204)	:WGEN<w>:FUNCtion:SQUare:DCYCle? (see page 1204)	<percent> ::= duty cycle percentage from 20% to 80% in NR1 format <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:MODulation:AM:DEPTh <percent> (see page 1205)	:WGEN<w>:MODulation:AM:DEPTh? (see page 1205)	<percent> ::= AM depth percentage from 0% to 100% in NR1 format <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:AM:FREQuency <frequency> (see page 1206)	:WGEN<w>:MODulation:AM:FREQuency? (see page 1206)	<frequency> ::= modulating waveform frequency in Hz in NR3 format <w> ::= 1 in NR1 format

Table 65 :WGEN<w> Commands Summary (continued)

Command	Query	Options and Query Returns
:WGEN<w>:MODulation:FM:DEViation <frequency> (see page 1207)	:WGEN<w>:MODulation:FM:DEViation? (see page 1207)	<frequency> ::= frequency deviation in Hz in NR3 format <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:FM:FREQuency <frequency> (see page 1208)	:WGEN<w>:MODulation:FM:FREQuency? (see page 1208)	<frequency> ::= modulating waveform frequency in Hz in NR3 format <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:FUNCtion <shape> (see page 1209)	:WGEN<w>:MODulation:FUNCtion? (see page 1209)	<shape> ::= {SINusoid} <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:NOISe <percent> (see page 1210)	:WGEN<w>:MODulation:NOISe? (see page 1210)	<percent> ::= 0 to 100 <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:STATe {{0 | OFF} | {1 | ON}} (see page 1211)	:WGEN<w>:MODulation:STATe? (see page 1211)	{0 | 1} <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:TYPE <type> (see page 1212)	:WGEN<w>:MODulation:TYPE? (see page 1212)	<type> ::= {AM | FM} <w> ::= 1 in NR1 format
:WGEN<w>:OUTPut {{0 | OFF} | {1 | ON}} (see page 1213)	:WGEN<w>:OUTPut? (see page 1213)	{0 | 1} <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:OUTPut:LOAD <impedance> (see page 1214)	:WGEN<w>:OUTPut:LOAD? (see page 1214)	<impedance> ::= {ONEMeg | FIFTy} <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:OUTPut:POLarity <polarity> (see page 1215)	:WGEN<w>:OUTPut:POLarity? (see page 1215)	<polarity> ::= {NORMal | INVerted} <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:PERiod <period> (see page 1216)	:WGEN<w>:PERiod? (see page 1216)	<period> ::= period in seconds in NR3 format <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:RST (see page 1217)	n/a	<w> ::= 1 to (# WaveGen outputs) in NR1 format

Table 65 :WGEN<w> Commands Summary (continued)

Command	Query	Options and Query Returns
:WGEN<w>:VOLTage <amplitude> (see page 1218)	:WGEN<w>:VOLTage? (see page 1218)	<amplitude> ::= amplitude in volts in NR3 format <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:VOLTage:HIGH <high> (see page 1219)	:WGEN<w>:VOLTage:HIGH? (see page 1219)	<high> ::= high-level voltage in volts, in NR3 format <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:VOLTage:LOW <low> (see page 1220)	:WGEN<w>:VOLTage:LOW? (see page 1220)	<low> ::= low-level voltage in volts, in NR3 format <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:VOLTage:OFFSet <offset> (see page 1221)	:WGEN<w>:VOLTage:OFFSet? (see page 1221)	<offset> ::= offset in volts in NR3 format <w> ::= 1 to (# WaveGen outputs) in NR1 format

Table 66 :WMEMory<r> Commands Summary

Command	Query	Options and Query Returns
:WMEMory<r>:CLEar (see page 1225)	n/a	<r> ::= 1 to (# ref waveforms) in NR1 format
:WMEMory<r>:DISPlay {{0 | OFF} | {1 | ON}} (see page 1226)	:WMEMory<r>:DISPlay? (see page 1226)	<r> ::= 1 to (# ref waveforms) in NR1 format {0 | 1}
:WMEMory<r>:LABel <string> (see page 1227)	:WMEMory<r>:LABel? (see page 1227)	<r> ::= 1 to (# ref waveforms) in NR1 format <string> ::= any series of 32 or less ASCII characters enclosed in quotation marks

Table 66 :WMEMory<r> Commands Summary (continued)

Command	Query	Options and Query Returns
:WMEMory<r>:SAVE <source> (see page 1228)	n/a	<r> ::= 1 to (# ref waveforms) in NR1 format <source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format NOTE: Math functions whose x-axis is not frequency can be saved as reference waveforms.
:WMEMory<r>:SKEW <skew> (see page 1229)	:WMEMory<r>:SKEW? (see page 1229)	<r> ::= 1 to (# ref waveforms) in NR1 format <skew> ::= time in seconds in NR3 format
:WMEMory<r>:YOFFset <offset>[suffix] (see page 1230)	:WMEMory<r>:YOFFset? (see page 1230)	<r> ::= 1 to (# ref waveforms) in NR1 format <offset> ::= vertical offset value in NR3 format [suffix] ::= {V | mV}
:WMEMory<r>:YRANge <range> (see page 1231)	:WMEMory<r>:YRANge? (see page 1231)	<r> ::= 1 to (# ref waveforms) in NR1 format <range> ::= vertical full-scale range value in NR3 format
:WMEMory<r>:YSCale <scale> (see page 1232)	:WMEMory<r>:YSCale? (see page 1232)	<r> ::= 1 to (# ref waveforms) in NR1 format <scale> ::= vertical units per division value in NR3 format

Syntax Elements

Number Format

NR1 specifies integer data.

NR3 specifies exponential data in floating point format (for example, -1.0E-3).

<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.

[ ] (Optional Syntax Terms)

Items enclosed in square brackets, [ ], are optional.

{ } (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.

::= (Defined As)

::= means "defined as".

For example, <A> ::= <B> indicates that <A> can be replaced by <B> in any statement containing <A>.

< > (Angle Brackets)

< > Angle brackets enclose words or characters that symbolize a program code parameter or an interface command.

... (Ellipsis)

... An ellipsis (trailing dots) indicates that the preceding element may be repeated one or more times.

n,..,p (Value Ranges)

n,..,p ::= all integers between n and p inclusive.

d (Digits)

d ::= A single ASCII numeric character 0 - 9.

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 Keysight 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.

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

6 Common (*) Commands

Commands defined by IEEE 488.2 standard that are common to all instruments.
See "Introduction to Common (*) Commands" on page 182.

Table 67 Common (*) Commands Summary

Command	Query	Options and Query Returns
*CLS (see page 184)	n/a	n/a
*ESE <mask> (see page 185)	*ESE? (see page 186)	<mask> ::= 0 to 255; an integer in NR1 format: Bit Weight Name Enables --- ------ ---- ---------- 7 128 PON Power On 6 64 URQ User Request 5 32 CME Command Error 4 16 EXE Execution Error 3 8 DDE Dev. Dependent Error 2 4 QYE Query Error 1 2 RQL Request Control 0 1 OPC Operation Complete
n/a	*ESR? (see page 187)	<status> ::= 0 to 255; an integer in NR1 format
n/a	*IDN? (see page 187)	KEYSIGHT 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
n/a	*LRN? (see page 190)	<learn_string> ::= current instrument setup as a block of data in IEEE 488.2 # format

Table 67 Common (*) Commands Summary (continued)

Command	Query	Options and Query Returns
*OPC (see page 191)	*OPC? (see page 191)	ASCII "1" is placed in the output queue when all pending device operations have completed.
n/a	*OPT? (see page 192)	<return_value> ::= 0,0,<license info> <license info> ::= <All field>, <reserved>, <Enhanced Security>, <Waveform Generator>, <Automotive Serial>, <reserved>, <Embedded Serial>, <reserved>, <reserved>, <reserved>, <reserved>, <reserved>, Memory 20M>, <Memory 50M>, <Memory 100M>, <Bandwidth>, <reserved>, <reserved>, <reserved>, <Demo Partner>, <Demo Keysight>
n/a	*OPT? (see page 192) (cont'd)	<All field> ::= {0 | All(d)} <reserved> ::= 0 <Enhanced Security> ::= {0 | SECA} <Waveform Generator> ::= {0 | WAVEGEN} <Automotive Serial> ::= {0 | HD300AUTA} <Embedded Serial> ::= {0 | HD300EMBA} <Memory 20M> ::= {0 | MEMUP20} <Memory 50M> ::= {0 | MEMUP50} <Memory 100M> ::= {0 | MEMUP100} <Bandwidth> ::= {BW200 | BW350 | BW500 | BW1000} <Demo Partner> ::= {0 | DIS} <Demo Keysight> ::= {0 | D24}

Table 67 Common (*) Commands Summary (continued)

Command	Query	Options and Query Returns
*RCL {<internal_fname> | <external_fname>} (see page 193)	n/a	<internal_fname> ::= quoted ASCII file name string beginning with "/User Files/" <external_fname> ::= quoted ASCII file name string beginning with "/USB1/" (or "/usb/"), "/USB2/", or "/USB3/" depending on the number of USB storage devices connected
*RST (see page 194)	n/a	See *RST (Reset) (see page 194)
*SAV {<internal_fname> | <external_fname>} (see page 197)	n/a	<internal_fname> ::= quoted ASCII file name string beginning with "/User Files/" <external_fname> ::= quoted ASCII file name string beginning with "/USB1/" (or "/usb/"), "/USB2/", or "/USB3/" depending on the number of USB storage devices connected
*SRE <mask> (see page 198)	*SRE? (see page 199)	<mask> ::= sum of all bits that are set, 0 to 255; an integer in NR1 format. <mask> ::= following values: Bit Weight Name Enables --- ------ ---- ---------- 7 128 OPER Operation Status Reg 6 64 ---- (Not used.) 5 32 ESB Event Status Bit 4 16 MAV Message Available 3 8 ---- (Not used.) 2 4 MSG Message 1 2 USR User 0 1 TRG Trigger

Table 67 Common (*) Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	*STB? (see page 200)	<value> ::= 0 to 255; an integer in NR1 format, as shown in the following: Bit Weight Name "1" Indicates --- ------ ---- --------------- 7 128 OPER Operation status condition occurred. 6 64 RQS/ MSS Instrument is requesting service. 5 32 ESB Enabled event status condition occurred. 4 16 MAV Message available. 3 8 ---- (Not used.) 2 4 MSG Message displayed. 1 2 USR User event condition occurred. 0 1 TRG A trigger occurred.
*TRG (see page 203)	n/a	n/a
n/a	*TST? (see page 204)	<result> ::= 0 or non-zero value; an integer in NR1 format
*WAI (see page 205)	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.

*CLS (Clear Status)

 (see page 1292)

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" on page 182
- "*STB? (Read Status Byte)" on page 200
- "*ESE (Standard Event Status Enable)" on page 185
- "*ESR? (Standard Event Status Register)" on page 187
- "*SRE (Service Request Enable)" on page 198
- ":SYSTem:ERRor[:NEXT]?" on page 1032
- ":STATus:PRESet" on page 887

*ESE (Standard Event Status Enable)

(see page 1292)

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.

Standard Event Status Enable Register

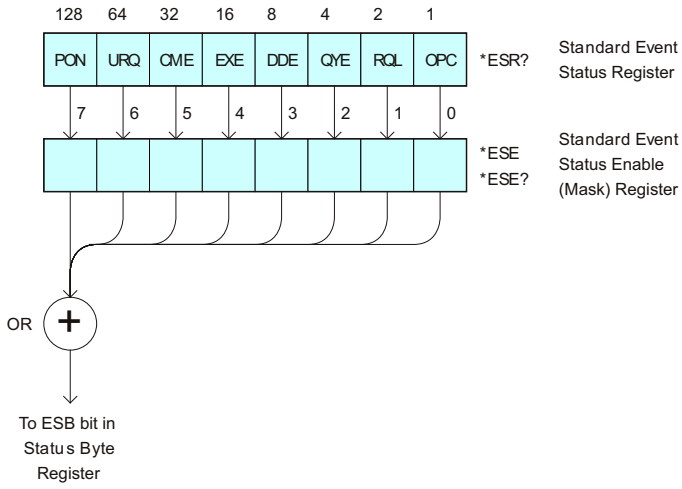


Table 68 Standard Event Status Enable (ESE) Register Bits

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.

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.

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" on page 182
- "*ESR? (Standard Event Status Register)" on page 187
- "*OPC (Operation Complete)" on page 191
- "*CLS (Clear Status)" on page 184

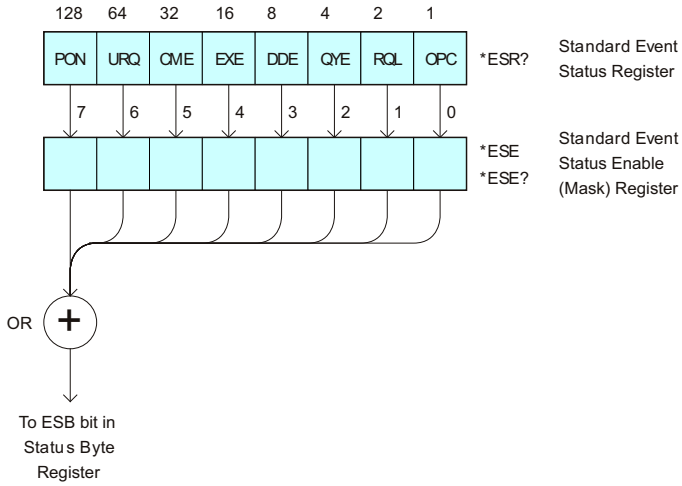
*ESR? (Standard Event Status Register)

(see page 1292)

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.

Standard Event Status Register



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

Table 69 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.

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.

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.

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" on page 182
- "*ESE (Standard Event Status Enable)" on page 185
- "*OPC (Operation Complete)" on page 191
- "*CLS (Clear Status)" on page 184
- ":SYSTem:ERRor[:NEXT]?" on page 1032

*IDN? (Identification Number)

C (see page 1292)

Query Syntax

```
*IDN?
```

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

Return Format

```
<manufacturer_string>,<model>,<serial_number>,X.XX.XX <NL>
<manufacturer_string> ::= KEYSIGHT TECHNOLOGIES
<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" on page 182
- "*OPT? (Option Identification)" on page 192
- ":SYSTem:PERSona[:MANufacturer]" on page 1040
- ":SYSTem:PERSona[:MANufacturer]:DEFault" on page 1041

*LRN? (Learn Device Setup)

(see page 1292)

Query Syntax

```
*LRN?
```

The *LRN? query result contains the current state of the instrument. This query is similar to the :SYSTem:SETup? (see page 1055) 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 has changed from previous Keysight 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" on page 182
- "*RCL (Recall)" on page 193
- "*SAV (Save)" on page 197
- ":SYSTem:SETup" on page 1055

*OPC (Operation Complete)

C (see page 1292)

Command Syntax

```
*OPC
```

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

You can use the *ESR? query to look at the OPC bit (bit 0) in the Standard Event Status Register to determine when an operation is complete.

NOTE

The front-panel graphical user interface can disable most of the remote interface, including the *OPC syntax, in certain situations. Bit 4 in the Operation Status Register shows whether the remote user interface is enabled or disabled. For more information, see ":STATus:OPERation:CONDition?" on page 898.

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.

The *OPC? query can be used between overlapped commands to make sure one is complete before the next one begins. The *OPC? query can also be used to pause a controller program until an operation is complete.

Return Format

```
<complete><NL>

<complete> ::= 1
```

See Also

- "Introduction to Common (*) Commands" on page 182
- "*ESE (Standard Event Status Enable)" on page 185
- "*ESR? (Standard Event Status Register)" on page 187
- "*CLS (Clear Status)" on page 184
- ":STATus:OPERation:CONDition?" on page 898
- ":STATus:OPERation[:EVENt]?" on page 902
- Chapter 4, “Sequential (Blocking) vs. Overlapped Commands,” starting on page 71
- "Synchronization with an Averaging Acquisition" on page 1277
- "Set Up the Oscilloscope" on page 1270
- "Example: Blocking and Polling Synchronization" on page 1279
- "Example: Waiting for IO Operation Complete" on page 1266

*OPT? (Option Identification)

(see page 1292)

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>, <Enhanced Security>,
   <Waveform Generator>, <Automotive Serial>, <reserved>,
   <Embedded Serial>, <reserved>, <reserved>, <reserved>,
   <reserved>, <reserved>, <Memory 20M>, <Memory 50M>,
   <Memory 100M>, <Bandwidth>, <reserved>, <reserved>, <reserved>,
   <Demo Partner>, <Demo Keysight>

<All field> ::= {0 | All(d)}

<reserved> ::= 0

<Enhanced Security> ::= {0 | SECA}

<Waveform Generator> ::= {0 | WAVEGEN}

<Automotive Serial> ::= {0 | HD300AUTA}

<Embedded Serial> ::= {0 | HD300EMBA}

<Memory 20M> ::= {0 | MEMUP20}

<Memory 50M> ::= {0 | MEMUP50}

<Memory 100M> ::= {0 | MEMUP100}

<Bandwidth> ::= {BW200 | BW350 | BW500 | BW1000}

<Demo Partner> ::= {0 | DIS}

<Demo Keysight> ::= {0 | D24}
```

The *OPT? query returns the following:

Module	Module Id
No modules attached	0,0,All(d),0,SECA*,WAVEGEN,HD300AUTA,HD300AERA,HD300EMBA,0,0,0,0,0,MEMUP20,MEMUP50*,MEMUP100,BW1000,0,0,0,DIS*,D24*

See Also

- "Introduction to Common (*) Commands" on page 182
- "*IDN? (Identification Number)" on page 189

*RCL (Recall)

(see page 1292)

Command Syntax

```
*RCL {<internal_fname> | <external_fname>}

<internal_fname> ::= quoted ASCII file name string beginning
   with "/User Files/"

<external_fname> ::= quoted ASCII file name string beginning
   with "/USB1/" (or "/usb/"), "/USB2/", or "/USB3/" depending
   on the number of USB storage devices connected
```

The *RCL command recalls an oscilloscope setup from a file.

See Also

- "Introduction to Common (*) Commands" on page 182
- "*SAV (Save)" on page 197
- ":RECall:SETup[:STARt]" on page 684

*RST (Reset)

 (see page 1292)

Command Syntax `*RST`

The *RST command places the instrument in a known state. This is the same as selecting **Factory Default** in the graphical user interface's Default dialog box (**Control > Default...**).

When you perform a factory default setup, there are no user settings that remain unchanged. To perform the equivalent of the front panel's **[Default Setup]** key, where some user settings (like preferences) remain unchanged, use the :SYSTem:PRESet command.

Reset conditions are:

Acquire Settings	
Mode	Normal
Averaging	Off
# Averages	8

Analog Channel Settings	
Channel 1	On
Channel 2/3/4	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 Settings	
Source	Channel 1

Digital Channel Settings	
Channel 0 - 15	Off
Labels	Off
Threshold	TTL (1.4 V)

Display Settings	
Persistence	Off
Grid	20%

Measurement Settings	
Source	Channel 1

Run Control Settings	
	Scope is running

Time Base Settings	
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 Settings	
Type	Edge
Mode	Auto
Coupling	dc

Trigger Settings	
Source	Channel 1
Level	0.0 V
Slope	Positive
HF Reject and noise reject	Off
Holdoff	40 ns
External probe attenuation	1:1
External Units	Volts
External Impedance	1 M Ohm (cannot be changed)

See Also
- "Introduction to Common (*) Commands" on page 182
- ":SYSTem:PRESet" on page 1044

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.
```

See complete example programs at: Chapter 44, “Programming Examples,” starting on page 1301

*SAV (Save)

 (see page 1292)

Command Syntax

```
*SAV {<internal_fname> | <external_fname>}

<internal_fname> ::= quoted ASCII file name string beginning
   with "/User Files/"

<external_fname> ::= quoted ASCII file name string beginning
   with "/USB1/" (or "/usb/"), "/USB2/", or "/USB3/" depending
   on the number of USB storage devices connected
```

The *SAV command saves the oscilloscope setup to a file.

See Also

- "Introduction to Common (*) Commands" on page 182
- "*RCL (Recall)" on page 193
- ":SAVE:SETup[:STARt]" on page 707

*SRE (Service Request Enable)

(see page 1292)

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

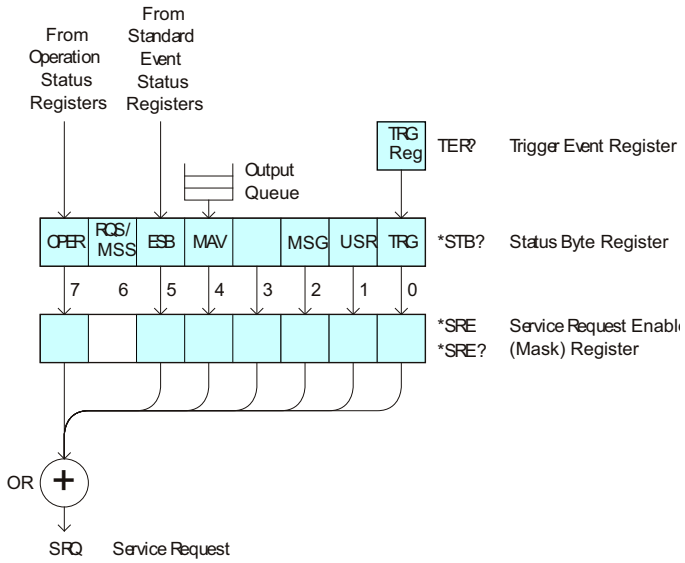


Table 70 Service Request Enable Register (SRE) Bits

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.

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).

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" on page 182
- "*STB? (Read Status Byte)" on page 200
- "*CLS (Clear Status)" on page 184

*STB? (Read Status Byte)

(see page 1292)

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.

Status Byte Register

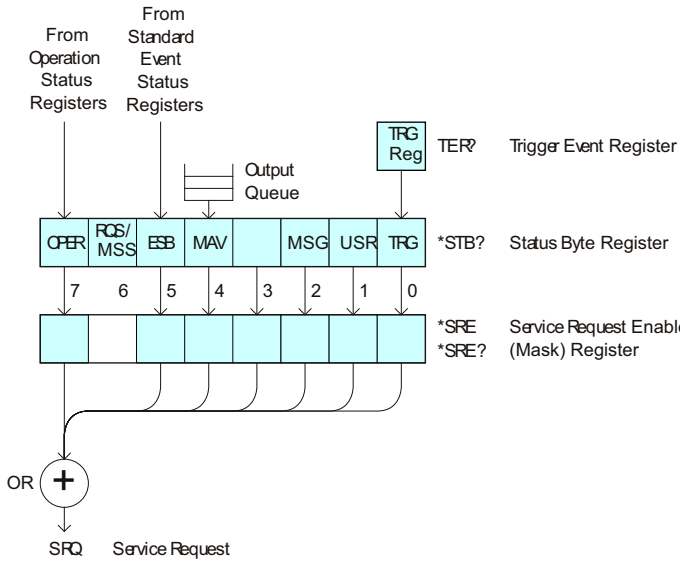


Table 71 Status Byte Register (STB) Bits

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.

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 programming interface 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.

Return Format

```
<value><NL>

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

See Also

- "Introduction to Common (*) Commands" on page 182
- "*SRE (Service Request Enable)" on page 198

*TRG (Trigger)

 (see page 1292)

Command Syntax `*TRG`

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

See Also

- "Introduction to Common (*) Commands" on page 182
- ":DIGitize" on page 218
- ":RUN" on page 221
- ":STOP" on page 225

*TST? (Self Test)

(see page 1292)

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" on page 182

*WAI (Wait To Continue)

 (see page 1292)

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" on page 182

7 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" on page 209.

Table 72 Root (:) Commands Summary

Command	Query	Options and Query Returns
:ACTivity (see page 210)	:ACTivity? (see page 210)	<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? (see page 211)	{0 | 1}; an integer in NR1 format
:AUToscale [<source>[,..,<source>]] (see page 212)	n/a	<source> ::= {CHANnel<n> | DIGital<d> | POD1 | POD2} <source> can be repeated up to 5 times <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:AUToscale:AMODe <value> (see page 214)	:AUToscale:AMODe? (see page 214)	<value> ::= {NORMal | CURRent}}
:AUToscale:CHANnels <value> (see page 215)	:AUToscale:CHANnels? (see page 215)	<value> ::= {ALL | DISPlayed}}
:AUToscale:FDEBug {{0 | OFF} | {1 | ON}} (see page 216)	:AUToscale:FDEBug? (see page 216)	{0 | 1}

Table 72 Root (:) Commands Summary (continued)

Command	Query	Options and Query Returns
:BLANk [<source>] (see page 217)	n/a	<source> ::= {CHANnel<n> | DIGital<d> | POD{1 | 2} | BUS{1 | 2} | FUNCtion<m> | MATH<m> | FFT | SBUS{1 | 2} | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format
:DIGitize [<source>[,..,<source>]] (see page 218)	n/a	<source> ::= {CHANnel<n> | DIGital<d> | POD{1 | 2} | BUS{1 | 2} | FUNCtion<m> | MATH<m> | FFT | SBUS{1 | 2}} <source> can be repeated up to 5 times <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <m> ::= 1 to (# math functions) in NR1 format
n/a	:RSTate? (see page 220)	n/a
:RUN (see page 221)	n/a	n/a
n/a	:SERial? (see page 222)	<return value> ::= quoted string containing serial number
:SINGle (see page 223)	n/a	n/a

Table 72 Root (:) Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:STATus? <display> (see page 224)	{0 | 1} <display> ::= {CHANnel<n> | DIGital<d> | POD{1 | 2} | BUS{1 | 2} | FUNCtion<m> | MATH<m> | FFT | SBUS{1 | 2} | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format
:STOP (see page 225)	n/a	n/a
n/a	:TER? (see page 226)	{0 | 1}
:VIEW <source> (see page 227)	n/a	<source> ::= {CHANnel<n> | DIGital<d> | POD{1 | 2} | BUS{1 | 2} | FUNCtion<m> | MATH<m> | FFT | SBUS{1 | 2} | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) 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

(see page 1292)

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" on page 209
- ":POD<n>:THReshold" on page 673
- ":DIGital<d>:THReshold" on page 318

:AER? (Arm Event Register)

(see page 1292)

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).

NOTE The :AER? query is an alias for the :STATus:OPERation:ARM:BIT0[:EVENt]? and :STATus:OPERation:ARM[:EVENt]? queries.

Return Format

```
<value><NL>

<value> ::= {0 | 1}; an integer in NR1 format.
```

See Also

- "Introduction to Root (:) Commands" on page 209
- ":STATus:OPERation:ENABle" on page 899
- ":STATus:OPERation:CONDition?" on page 898
- ":STATus:OPERation[:EVENt]?" on page 902
- "*STB? (Read Status Byte)" on page 200
- "*SRE (Service Request Enable)" on page 198
- ":STATus:OPERation:ARM:BIT<b>[:EVENt]?" on page 910
- ":STATus:OPERation:ARM[:EVENt]?" on page 915

:AUToscale

(see page 1292)

Command Syntax

```
:AUToscale

:AUToscale [<source>[,..,<source>]]

<source> ::= {DIGital<d> | POD1 | POD2 | CHANnel<n>}

<n> ::= 1 to (# analog channels) in NR1 format

<d> ::= 0 to (# digital channels - 1) in NR1 format

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 **[Auto Scale]** 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" on page 215) 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.
- Math waveforms.
- Reference waveforms.
- Zoomed (delayed) time base mode.

For further information on :AUToscale, see the *User's Guide*.

See Also

- "Introduction to Root (:) Commands" on page 209

- ":AUToscale:CHANnels" on page 215
- ":AUToscale:AMODe" on page 214

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 Auto Scale key.
```

See complete example programs at: Chapter 44, “Programming Examples,” starting on page 1301

:AUToscale:AMODe

(see page 1292)

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" on page 209
- ":AUToscale" on page 212
- ":AUToscale:CHANnels" on page 215
- ":ACQuire:TYPE" on page 247
- ":ACQuire:MODE" on page 237

:AUToscale:CHANnels

(see page 1292)

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" on page 209
- ":AUToscale" on page 212
- ":AUToscale:AMODe" on page 214
- ":VIEW" on page 227
- ":BLANk" on page 217

:AUToscale:FDEBug

 (see page 1292)

Command Syntax

```
:AUToscale:FDEBug <on_off>

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

The :AUToscale:FDEBug command turns fast debug auto scaling on or off.

The Fast Debug option changes the behavior of :AUToscale to let you make quick visual comparisons to determine whether the signal being probed is a DC voltage, ground, or an active AC signal.

Channel coupling is maintained for easy viewing of oscillating signals.

Query Syntax

```
:AUToscale:FDEBug?
```

The :AUToscale:FDEBug? query returns the current autoscale fast debug setting.

Return Format

```
<on_off><NL>

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

See Also

- "Introduction to Root (:) Commands" on page 209
- ":AUToscale" on page 212

:BLANk

(see page 1292)

Command Syntax

```
:BLANk [<source>]

<source> ::= {CHANnel<n> | DIGital<d> | POD{1 | 2}
             | BUS{1 | 2} | FUNCtion<m> | MATH<m> | FFT | SBUS{1 | 2}
             | WMEMory<r>}

<n> ::= 1 to (# analog channels) in NR1 format

<d> ::= 0 to (# digital channels - 1) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :BLANk command turns off (stops displaying) the specified channel, digital pod, math function, serial decode bus, or reference waveform location. 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, :DIGital<n>:DISPlay, :SBUS<n>:DISPlay, or :WMEMory<r>:DISPlay, are the preferred method to turn on/off a channel, etc.

NOTE MATH<m> is an alias for FUNCtion<m>.

See Also

- "Introduction to Root (:) Commands" on page 209
- ":DISPlay:CLEar" on page 334
- ":CHANnel<n>:DISPlay" on page 275
- ":DIGital<d>:DISPlay" on page 314
- ":FUNCtion<m>:DISPlay" on page 431
- ":POD<n>:DISPlay" on page 669
- ":SBUS<n>:DISPlay" on page 718
- ":WMEMory<r>:DISPlay" on page 1226
- ":STATus?" on page 224
- ":VIEW" on page 227

Example Code

- "Example Code" on page 227

:DIGitize

(see page 1292)

Command Syntax

```
:DIGitize [<source>[,..,<source>]]

<source> ::= {CHANnel<n> | DIGital<d> | POD{1 | 2}
             | BUS{1 | 2} | FUNCtion<m> | MATH<m> | FFT | SBUS{1 | 2}}

<n> ::= 1 to (# analog channels) in NR1 format

<d> ::= 0 to (# digital channels - 1) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

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 The :DIGitize command is only executed when the :TIMebase:MODE is MAIN or WINDow.

NOTE To halt a :DIGitize in progress, use the device clear command.

NOTE MATH<m> is an alias for FUNCtion<m>.

See Also

- "Introduction to Root (:) Commands" on page 209
- ":RUN" on page 221
- ":SINGle" on page 223
- ":STOP" on page 225
- ":TIMebase:MODE" on page 1070
- Chapter 8, ":ACQuire Commands," starting on page 229
- Chapter 36, ":WAVeform Commands," starting on page 1143
- Chapter 4, "Sequential (Blocking) vs. Overlapped Commands," starting on page 71
- "Example: Checking for Armed Status" on page 1261

Example Code

```
' Capture an acquisition using :DIGitize.
' ------------------------------------------------------------------
myScope.WriteString ":DIGitize CHANnel1"
```

See complete example programs at: Chapter 44, “Programming Examples,” starting on page 1301

:RSTate?

N (see page 1292)

Query Syntax `:RSTate?`

The :RSTate? query returns the run state:

- RUN – The oscilloscope is acquiring and displaying new waveforms.
- STOP – The oscilloscope is no longer acquiring new waveforms.
- SING – A single acquisition has been started and the oscilloscope is waiting for the trigger condition to be met.

These are the same run states displayed on the front panel and in the user interface.

Return Format `{RUN | STOP | SING}<NL>`

See Also

- ":RUN" on page 221
- ":SINGle" on page 223
- ":STOP" on page 225
- ":STATus:OPERation[:EVENt]?" on page 902

:RUN

C (see page 1292)

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" on page 209
- ":SINGle" on page 223
- ":STOP" on page 225

Example Code

```
' RUN_STOP - (not executed in this example)
'   - RUN starts the data acquisition 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.
```

See complete example programs at: Chapter 44, “Programming Examples,” starting on page 1301

:SERial?

N (see page 1292)

Query Syntax `:SERial?`

The :SERial? query returns the serial number of the instrument.

Return Format: `Quoted string<NL>`

See Also

- "Introduction to Root (:) Commands" on page 209
- "Commands Not Supported in Multiple Program Message Units" on page 1297

:SINGle

 (see page 1292)

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" on page 209
- ":RUN" on page 221
- ":STOP" on page 225

:STATus?

(see page 1292)

Query Syntax

```
:STATus? <source>

<source> ::= {CHANnel<n> | DIGital<d> | POD{1 | 2}
             | BUS{1 | 2} | FUNCtion<m> | MATH<m> | FFT | SBUS{1 | 2}
             | WMEMory<r>}

<n> ::= 1 to (# analog channels) in NR1 format

<d> ::= 0 to (# digital channels - 1) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :STATus? query reports whether the channel, function, serial decode bus, or reference waveform location specified by <source> is displayed.

NOTE MATH<m> is an alias for FUNCtion<m>.

Return Format

```
<value><NL>

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

See Also

- "Introduction to Root (:) Commands" on page 209
- ":BLANk" on page 217
- ":CHANnel<n>:DISPlay" on page 275
- ":DIGital<d>:DISPlay" on page 314
- ":FUNCtion<m>:DISPlay" on page 431
- ":POD<n>:DISPlay" on page 669
- ":SBUS<n>:DISPlay" on page 718
- ":WMEMory<r>:DISPlay" on page 1226
- ":VIEW" on page 227

:STOP

 (see page 1292)

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" on page 209
- ":RUN" on page 221
- ":SINGle" on page 223

Example Code

- "Example Code" on page 221

:TER? (Trigger Event Register)

(see page 1292)

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.

NOTE **The :TER? query is an alias for the :STATus:TRIGger:BIT0[:EVENt]? and :STATus:TRIGger[:EVENt]? queries.**

Return Format

```
<value><NL>

<value> ::= {1 | 0}; a 16-bit integer in NR1 format.
```

See Also

- "Introduction to Root (:) Commands" on page 209
- "*SRE (Service Request Enable)" on page 198
- "*STB? (Read Status Byte)" on page 200
- ":STATus:TRIGger:BIT<b>[:EVENt]?" on page 1001
- ":STATus:TRIGger[:EVENt]?" on page 1006

:VIEW

N (see page 1292)

Command Syntax

```
:VIEW <source>

<source> ::= {CHANnel<n> | DIGital<d> | POD{1 | 2}
             | BUS{1 | 2} | FUNCtion<m> | MATH<m> | FFT | SBUS{1 | 2}
             | WMEMory<r>}

<n> ::= 1 to (# analog channels) in NR1 format

<d> ::= 0 to (# digital channels - 1) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :VIEW command turns on the specified channel, function, serial decode bus, or reference waveform location.

NOTE MATH<m> is an alias for FUNCtion<m>.

See Also

- "Introduction to Root (:) Commands" on page 209
- ":BLANk" on page 217
- ":CHANnel<n>:DISPlay" on page 275
- ":DIGital<d>:DISPlay" on page 314
- ":FUNCtion<m>:DISPlay" on page 431
- ":POD<n>:DISPlay" on page 669
- ":SBUS<n>:DISPlay" on page 718
- ":WMEMory<r>:DISPlay" on page 1226
- ":STATus?" on page 224

Example Code

```
' VIEW_BLANK - (not executed in this example)
'  - VIEW turns on (starts displaying) a channel.
'  - BLANK turns off (stops displaying) a channel.
' myScope.WriteString ":BLANk CHANnel1"    ' Turn channel 1 off.
' myScope.WriteString ":VIEW CHANnel1"    ' Turn channel 1 on.
```

See complete example programs at: Chapter 44, “Programming Examples,” starting on page 1301

8 :ACQuire Commands

Set the parameters for acquiring and storing data. See "Introduction to :ACQuire Commands" on page 230.

Table 73 :ACQuire Commands Summary

Command	Query	Options and Query Returns
:ACQuire:BANDwidth <limit> (see page 232)	:ACQuire:BANDwidth? (see page 232)	<limit> ::= {OFF | <global_bw_limit>} in NR3 format
:ACQuire:COMPlete <complete> (see page 233)	:ACQuire:COMPlete? (see page 233)	<complete> ::= 100; an integer in NR1 format
:ACQuire:COUNt <count> (see page 234)	:ACQuire:COUNt? (see page 234)	<count> ::= an integer from 2 to 65536 in NR1 format
:ACQuire:DIGitizer {{0 | OFF} | {1 | ON}} (see page 235)	:ACQuire:DIGitizer? (see page 235)	{0 | 1}
:ACQuire:MANual {{0 | OFF} | {1 | ON}} (see page 236)	:ACQuire:MANual? (see page 236)	{0 | 1}
:ACQuire:MODE <mode> (see page 237)	:ACQuire:MODE? (see page 237)	<mode> ::= {RTIMe | ETIMe | SEGMented}
:ACQuire:POINts[:ANALog] <points> (see page 238)	:ACQuire:POINts[:ANALog]? (see page 238)	<points> ::= {AUTO | <points_value>} <points_value> ::= desired analog memory depth in integer NR1 format
:ACQuire:POINts[:ANALog]:AUTO {{0 | OFF} | {1 | ON}} (see page 239)	:ACQuire:POINts[:ANALog]:AUTO? (see page 239)	{0 | 1}
:ACQuire:SEGMented:ANALyze (see page 240)	n/a	n/a

Table 73 :ACQuire Commands Summary (continued)

Command	Query	Options and Query Returns
:ACQuire:SEGMented:COUNt <count> (see page 241)	:ACQuire:SEGMented:COUNt? (see page 241)	<count> ::= an integer from 2 to 1000 in NR1 format
:ACQuire:SEGMented:INDex <index> (see page 242)	:ACQuire:SEGMented:INDex? (see page 242)	<index> ::= an integer from 1 to 1000 in NR1 format
:ACQuire:SRATe[:ANALog] <rate> (see page 245)	:ACQuire:SRATe[:ANALog]? (see page 245)	<rate> ::= {AUTO | <sample_rate>} <sample_rate> ::= desired analog sample rate in NR3 format
:ACQuire:SRATe[:ANALog]:AUTO {{0 | OFF} | {1 | ON}} (see page 246)	:ACQuire:SRATe[:ANALog]:AUTO? (see page 246)	{0 | 1}
:ACQuire:TYPE <type> (see page 247)	:ACQuire:TYPE? (see page 247)	<type> ::= {NORMal | AVERage | 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.

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 2 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.

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;SEGM:COUN 2
```

:ACQuire:BANDwidth

N (see page 1292)

Command Syntax

```
:ACQuire:BANDwidth <limit>

<limit> ::= {OFF | <global_bw_limit>}

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

The :ACQuire:BANDwidth command lets you limit the bandwidth of all channels to remove unwanted high frequency noise from the waveform and potentially increase the number of bits of vertical resolution. You can select these bandwidths:

- 350000000 – 350 MHz / 14 Bits
- 200000000 – 200 MHz / 14 Bits
- 100000000 – 100 MHz / 15 Bits HD
- 50000000 – 50 MHz / 16 Bits HD
- 20000000 – 20 MHz / 16 Bits HD
- 10000000 – 10 MHz / 16 Bits HD
- 5000000 – 5 MHz / 16 Bits HD

The selections marked with "HD" (high-definition) improve the number of bits of vertical resolution.

Note that each channel also has an independent bandwidth limit option (see :CHANnel<n>:BWLimit). When the global bandwidth limit and an individual channel bandwidth limit are both selected, the lower bandwidth limit is applied.

Query Syntax

```
:ACQuire:BANDwidth?
```

The :ACQuire:BANDwidth? query returns the current global bandwidth limit setting.

Return Format

```
<limit><NL>

<limit> ::= {OFF | <global_bw_limit>}

<global_bw_limit> ::= {350000000 | 200000000 | 100000000 | 50000000
   | 20000000 | 10000000 | 5000000} in NR3 format
```

See Also

- ":CHANnel<n>:BWLimit" on page 273

:ACQuire:COMPlete

 (see page 1292)

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" on page 230
- ":ACQuire:TYPE" on page 247
- ":DIGitize" on page 218
- ":WAVeform:POINts" on page 1156

Example Code

```
' ACQUIRE_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"
```

See complete example programs at: Chapter 44, “Programming Examples,” starting on page 1301

:ACQuire:COUNt

(see page 1292)

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 2 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 2 to 65536 in NR1 format
```

See Also

- "Introduction to :ACQuire Commands" on page 230
- ":ACQuire:TYPE" on page 247
- ":DIGitize" on page 218
- ":WAVeform:COUNt?" on page 1152

:ACQuire:DIGitizer

N (see page 1292)

Command Syntax

```
:ACQuire:DIGitizer {{0 | OFF} | {1 | ON}}
```

The :ACQuire:DIGitizer command turns Digitizer mode on or off.

Normally, when Digitizer mode is disabled (Automatic mode), the oscilloscope's time per division setting determines the sample rate and memory depth so as to fill the waveform display with data while the oscilloscope is running (continuously making acquisitions). For single acquisitions, the time/division setting still determines the sample rate, but the maximum amount of acquisition memory is used.

In Digitizer mode, you choose the acquisition sample rate and memory depth, and those settings are used even though the captured data may extend way beyond the edges of, or take up just a small portion of, the waveform display, depending on the oscilloscope's time/div setting.

Digitizer mode cannot be used along with these other oscilloscope features: horizontal Zoom display, time references other than Center, segmented memory, serial decode, digital channels, frequency response analysis, mask test, and the power application. In most cases, enabling one of these features when Digitizer mode is enabled will automatically disable Digitizer mode, and then disabling the feature will automatically reenable Digitizer mode.

Digitizer mode primarily aids external software that controls and combines data from multiple instruments.

The :ACQuire:DIGitizer command is an alias for :ACQuire:MANual.

Query Syntax

```
:ACQuire:DIGitizer?
```

The :ACQuire:DIGitizer? query returns whether Digitizer mode is off or on.

Return Format

```
<setting><NL>

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

See Also

- ":ACQuire:MANual" on page 236
- ":ACQuire:SRATe[:ANALog]" on page 245
- ":ACQuire:SRATe[:ANALog]:AUTO" on page 246
- ":ACQuire:POINts[:ANALog]" on page 238
- ":ACQuire:POINts[:ANALog]:AUTO" on page 239

:ACQuire:MANual

N (see page 1292)

Command Syntax `:ACQuire:MANual {{0 | OFF} | {1 | ON}}`

The :ACQuire:MANual command enables or disables Automatic determination of the analog channel sample rate:

- OFF – the analog channel sample rate is automatically determined by the oscilloscope based on the horizontal time/div setting (Automatic mode, the oscilloscope's default).

  This is equivalent to turning Digitizer mode off using the :ACQuire:DIGitizer command.

- ON – the analog channel sample rate is set using the :ACQuire:SRATe[:ANALog] command.

  This is equivalent to turning Digitizer mode on using the :ACQuire:DIGitizer command.

The Automatic or Digitizer mode setting also affects the memory depth (see the :ACQuire:POINts[:ANALog] command).

Query Syntax `:ACQuire:MANual?`

The :ACQuire:MANual? query returns the automatic analog sample rate setting.

Return Format `<setting><NL>`

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

See Also

- ":ACQuire:DIGitizer" on page 235
- ":ACQuire:SRATe[:ANALog]" on page 245
- ":ACQuire:POINts[:ANALog]" on page 238

:ACQuire:MODE

(see page 1292)

Command Syntax

```
:ACQuire:MODE <mode>

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

The :ACQuire:MODE command sets the acquisition mode of the oscilloscope.

- The :ACQuire:MODE RTIMe command sets the oscilloscope in real time mode.

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

- The :ACQuire:MODE SEGMented command sets the oscilloscope in segmented memory mode.

Query Syntax

```
:ACQuire:MODE?
```

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

Return Format

```
<mode_argument><NL>

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

See Also

- "Introduction to :ACQuire Commands" on page 230
- ":ACQuire:TYPE" on page 247

:ACQuire:POINts[:ANALog]

(see page 1292)

Command Syntax

```
:ACQuire:POINts[:ANALog] <points>

<points> ::= {AUTO | <points_value>}

<points_value> ::= desired analog memory depth in integer NR1 format
```

Sets the desired acquisition memory depth.

- AUTO – puts the oscilloscope into Automatic (default) mode where the optimal memory depth is selected automatically by the oscilloscope.
- <points_value> – If a numerical memory depth is entered, this puts the oscilloscope into Digitizer (manual) Mode with the desired number of points.

The Automatic or Digitizer mode setting also affects the sample rate (see the :ACQuire:SRATe[:ANALog] command).

The actual memory depth used is returned by the :ACQuire:POINts[:ANALog]? query.

For :SINGle acquisitions, the requested memory depth is used.

When the oscilloscope is running (:RUN command, continuously making acquisitions) or when the :DIGitize command is used, the requested amount of memory is smaller.

Query Syntax

```
:ACQuire:POINts[:ANALog]?
```

The :ACQuire:POINts[:ANALog]? 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" on page 230
- ":ACQuire:SRATe[:ANALog]" on page 245
- ":ACQuire:POINts[:ANALog]:AUTO" on page 239
- ":ACQuire:MANual" on page 236
- ":ACQuire:DIGitizer" on page 235
- ":DIGitize" on page 218
- ":WAVeform:POINts" on page 1156

:ACQuire:POINts[:ANALog]:AUTO

N (see page 1292)

Command Syntax `:ACQuire:POINts[:ANALog]:AUTO {{0 | OFF} | {1 | ON}}`

The :ACQuire:POINts[:ANALog]:AUTO command enables or disables Automatic determination of the analog channel memory depth:

- ON – the analog channel memory depth is automatically determined by the oscilloscope based on the horizontal time/div setting (Automatic mode, the oscilloscope's default).

  This is equivalent to turning Digitizer mode off using the :ACQuire:MANual or :ACQuire:DIGitizer command.

- OFF – the analog channel memory depth is set using the :ACQuire:POINts[:ANALog] command.

  This is equivalent to turning Digitizer mode on using the :ACQuire:MANual or :ACQuire:DIGitizer command.

The Automatic or Digitizer mode setting also affects the sample rate (see the :ACQuire:SRATe[:ANALog] command).

Query Syntax `:ACQuire:POINts[:ANALog]:AUTO?`

The :ACQuire:POINts[:ANALog]:AUTO? query returns the automatic analog memory depth setting.

Return Format `<setting><NL>`

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

See Also

- ":ACQuire:MANual" on page 236
- ":ACQuire:DIGitizer" on page 235
- ":ACQuire:POINts[:ANALog]" on page 238
- ":ACQuire:SRATe[:ANALog]" on page 245

:ACQuire:SEGMented:ANALyze

 (see page 1292)

Command Syntax `:ACQuire:SEGMented:ANALyze`

This command calculates measurement statistics and/or infinite persistence over all segments that have been acquired. It corresponds to the graphical user interface **Analyze Segments** button which appears in the Segmented dialog box.

In order to use this command, the oscilloscope must be stopped and in segmented acquisition mode, with either measurements, infinite persistence, histograms, or protocol listing on.

See Also

- ":ACQuire:MODE" on page 237
- ":ACQuire:SEGMented:COUNt" on page 241
- "Introduction to :ACQuire Commands" on page 230

:ACQuire:SEGMented:COUNt

N (see page 1292)

Command Syntax

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

<count> ::= an integer from 2 to 2000 in NR1 format
```

The :ACQuire:SEGMented:COUNt command sets the number of memory segments to acquire.

The segmented memory acquisition mode is enabled with the :ACQuire:MODE command, and data is acquired using the :DIGitize, :SINGle, or :RUN commands. The number of memory segments in the current acquisition is returned by the :WAVeform:SEGMented:COUNt? query.

The maximum number of segments may be limited by the memory depth of your oscilloscope.

Query Syntax

```
:ACQuire:SEGMented:COUNt?
```

The :ACQuire:SEGMented:COUNt? query returns the current count setting.

Return Format

```
<count><NL>

<count> ::= an integer from 2 to 2000 in NR1 format
```

See Also

- ":ACQuire:MODE" on page 237
- ":DIGitize" on page 218
- ":SINGle" on page 223
- ":RUN" on page 221
- ":WAVeform:SEGMented:COUNt?" on page 1164
- ":ACQuire:SEGMented:ANALyze" on page 240
- "Introduction to :ACQuire Commands" on page 230

Example Code

- "Example Code" on page 242

:ACQuire:SEGMented:INDex

 (see page 1292)

Command Syntax

```
:ACQuire:SEGMented:INDex <index>

<index> ::= an integer from 1 to 2000 in NR1 format
```

The :ACQuire:SEGMented:INDex command sets the index into the memory segments that have been acquired.

The segmented memory acquisition mode is enabled with the :ACQuire:MODE command. The number of segments to acquire is set using the :ACQuire:SEGMented:COUNt command, and data is acquired using the :DIGitize, :SINGle, or :RUN commands. The number of memory segments that have been acquired is returned by the :WAVeform:SEGMented:COUNt? query. The time tag of the currently indexed memory segment is returned by the :WAVeform:SEGMented:TTAG? query.

The maximum number of segments may be limited by the memory depth of your oscilloscope.

Query Syntax

```
:ACQuire:SEGMented:INDex?
```

The :ACQuire:SEGMented:INDex? query returns the current segmented memory index setting.

Return Format

```
<index><NL>

<index> ::= an integer from 1 to 2000 in NR1 format
```

See Also

- ":ACQuire:MODE" on page 237
- ":ACQuire:SEGMented:COUNt" on page 241
- ":DIGitize" on page 218
- ":SINGle" on page 223
- ":RUN" on page 221
- ":WAVeform:SEGMented:COUNt?" on page 1164
- ":WAVeform:SEGMented:TTAG?" on page 1165
- ":ACQuire:SEGMented:ANALyze" on page 240
- "Introduction to :ACQuire Commands" on page 230

Example Code

```
' Segmented memory commands example.
' ------------------------------------------------------------------

Option Explicit

Public myMgr As VisaComLib.ResourceManager
Public myScope As VisaComLib.FormattedIO488
Public varQueryResult As Variant
Public strQueryResult As String
```

```
#If VBA7 Then
    Private Declare PtrSafe Sub Sleep Lib "kernel32" (ByVal dwMillisecon
ds As LongPtr)
#Else
    Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Lo
ng)
#End If

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("USB0::0x2A8D::0x4704::Unset::0::INSTR")
  myScope.IO.Clear   ' Clear the interface.

  ' Turn on segmented memory acquisition mode.
  myScope.WriteString ":ACQuire:MODE SEGMented"
  myScope.WriteString ":ACQuire:MODE?"
  strQueryResult = myScope.ReadString
  Debug.Print "Acquisition mode: " + strQueryResult

  ' Set the number of segments to 25.
  myScope.WriteString ":ACQuire:SEGMented:COUNt 25"
  myScope.WriteString ":ACQuire:SEGMented:COUNt?"
  strQueryResult = myScope.ReadString
  Debug.Print "Acquisition memory segments: " + strQueryResult

  ' If data will be acquired within the IO timeout:
  myScope.IO.Timeout = 10000
  myScope.WriteString ":DIGitize"
  Debug.Print ":DIGitize blocks until all segments acquired."
  myScope.WriteString ":WAVeform:SEGMented:COUNt?"
  varQueryResult = myScope.ReadNumber

  ' Or, to poll until the desired number of segments acquired:
  myScope.WriteString ":SINGle"
  Debug.Print ":SINGle does not block until all segments acquired."
  Do
    Sleep 100   ' Small wait to prevent excessive queries.
    myScope.WriteString ":WAVeform:SEGMented:COUNt?"
    varQueryResult = myScope.ReadNumber
  Loop Until varQueryResult = 25

  Debug.Print "Number of segments in acquired data: " _
      + FormatNumber(varQueryResult)

  Dim lngSegments As Long
  lngSegments = varQueryResult

  ' For each segment:
  Dim dblTimeTag As Double
  Dim lngI As Long
```

```
  For lngI = lngSegments To 1 Step -1

    ' Set the segmented memory index.
    myScope.WriteString ":ACQuire:SEGMented:INDex " + CStr(lngI)
    myScope.WriteString ":ACQuire:SEGMented:INDex?"
    strQueryResult = myScope.ReadString
    Debug.Print "Acquisition memory segment index: " + strQueryResult

    ' Display the segment time tag.
    myScope.WriteString ":WAVeform:SEGMented:TTAG?"
    dblTimeTag = myScope.ReadNumber
    Debug.Print "Segment " + CStr(lngI) + " time tag: " _
        + FormatNumber(dblTimeTag, 12)

  Next lngI

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error:" + vbCrLf + Err.Description

End Sub
```

:ACQuire:SRATe[:ANALog]

N (see page 1292)

Command Syntax

```
:ACQuire:SRATe[:ANALog] <rate>

<rate> ::= {AUTO | <sample_rate>}

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

Sets the desired acquisition sample rate:

- AUTO – puts the oscilloscope into Automatic (default) mode where the optimal sample rate is selected automatically by the oscilloscope.
- <sample_rate> – If a numerical sample rate value is entered, this puts the oscilloscope into Digitizer (manual) mode with the desired sample rate.

The Automatic or Digitizer mode setting also affects the memory depth (see the :ACQuire:POINts[:ANALog] command).

The actual sample rate used is returned by the :ACQuire:SRATe[:ANALog]? query.

Query Syntax

```
:ACQuire:SRATe[:ANALog]? [MAXimum]
```

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

When the MAXimum parameter is used, the oscilloscope's maximum possible sample rate is returned.

Return Format

```
<sample_rate><NL>

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

See Also

- "Introduction to :ACQuire Commands" on page 230
- ":ACQuire:POINts[:ANALog]" on page 238
- ":ACQuire:SRATe[:ANALog]:AUTO" on page 246
- ":ACQuire:MANual" on page 236
- ":ACQuire:DIGitizer" on page 235

:ACQuire:SRATe[:ANALog]:AUTO

N (see page 1292)

Command Syntax `:ACQuire:SRATe[:ANALog]:AUTO {{0 | OFF} | {1 | ON}}`

The :ACQuire:SRATe[:ANALog]:AUTO command enables or disables Automatic determination of the analog channel sample rate:

- ON – the analog channel sample rate is automatically determined by the oscilloscope based on the horizontal time/div setting (Automatic mode, the oscilloscope's default).

  This is equivalent to turning Digitizer mode off using the :ACQuire:MANual or :ACQuire:DIGitizer command.

- OFF – the analog channel sample rate is set using the :ACQuire:SRATe[:ANALog] command.

  This is equivalent to turning Digitizer mode on using the :ACQuire:MANual or :ACQuire:DIGitizer command.

The Automatic or Digitizer mode setting also affects the memory depth (see the :ACQuire:POINts[:ANALog] command).

Query Syntax `:ACQuire:SRATe[:ANALog]:AUTO?`

The :ACQuire:SRATe[:ANALog]:AUTO? query returns the automatic analog sample rate setting.

Return Format `<setting><NL>`

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

See Also

- ":ACQuire:MANual" on page 236
- ":ACQuire:DIGitizer" on page 235
- ":ACQuire:SRATe[:ANALog]" on page 245
- ":ACQuire:POINts[:ANALog]" on page 238

:ACQuire:TYPE

 (see page 1292)

Command Syntax

```
:ACQuire:TYPE <type>

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

The :ACQuire:TYPE command selects the type of data acquisition that is to take place. The acquisition types are:

- NORMal – sets the oscilloscope in the normal mode.
- AVERage – 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 to 65536. The COUNt value determines the number of averages that must be acquired.

  The AVERage type is not available when in segmented memory mode (:ACQuire:MODE SEGMented).
- PEAK – sets the oscilloscope in the peak detect mode. In this mode, :ACQuire:COUNt has no meaning.

The AVERage type can give you extra bits of vertical resolution. See the *User's Guide* for an explanation. When getting waveform data acquired using the AVERage type, be sure to use the WORD or ASCii waveform data formats to get the extra bits of vertical resolution.

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 | PEAK}
```

See Also

- "Introduction to :ACQuire Commands" on page 230
- ":ACQuire:COUNt" on page 234
- ":ACQuire:MODE" on page 237
- ":DIGitize" on page 218
- ":WAVeform:FORMat" on page 1155
- ":WAVeform:TYPE?" on page 1172
- ":WAVeform:PREamble?" on page 1160

Example Code

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

See complete example programs at: Chapter 44, “Programming Examples,” starting on page 1301

9 :BUS<n> Commands

Control all oscilloscope functions associated with buses made up of digital channels. See "Introduction to :BUS<n> Commands" on page 250.

Table 74 :BUS<n> Commands Summary

Command	Query	Options and Query Returns
:BUS<n>:BIT<m> {{0 | OFF} | {1 | ON}} (see page 251)	:BUS<n>:BIT<m>? (see page 251)	{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}} (see page 252)	:BUS<n>:BITS? (see page 252)	<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 (see page 254)	n/a	<n> ::= 1 or 2; an integer in NR1 format
:BUS<n>:DISPlay {{0 | OFF} | {1 | ON}} (see page 255)	:BUS<n>:DISPlay? (see page 255)	{0 | 1} <n> ::= 1 or 2; an integer in NR1 format

Table 74 :BUS<n> Commands Summary (continued)

Command	Query	Options and Query Returns
:BUS<n>:LABel <string> (see page 256)	:BUS<n>:LABel? (see page 256)	<string> ::= quoted ASCII string up to 32 characters <n> ::= 1 or 2; an integer in NR1 format
:BUS<n>:MASK <mask> (see page 257)	:BUS<n>:MASK? (see page 257)	<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.

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>:BIT<m>

(see page 1292)

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.

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" on page 250
- ":BUS<n>:BITS" on page 252
- ":BUS<n>:CLEar" on page 254
- ":BUS<n>:DISPlay" on page 255
- ":BUS<n>:LABel" on page 256
- ":BUS<n>:MASK" on page 257

Example Code

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

:BUS<n>:BITS

N (see page 1292)

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.

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" on page 250
- ":BUS<n>:BIT<m>" on page 251
- ":BUS<n>:CLEar" on page 254
- ":BUS<n>:DISPlay" on page 255
- ":BUS<n>:LABel" on page 256
- ":BUS<n>:MASK" on page 257
- "Commands Not Supported in Multiple Program Message Units" on page 1297
- "Queries Not Supported in Multiple Program Message Units" on page 1297

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>:CLEar

(see page 1292)

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.

See Also

- "Introduction to :BUS<n> Commands" on page 250
- ":BUS<n>:BIT<m>" on page 251
- ":BUS<n>:BITS" on page 252
- ":BUS<n>:DISPlay" on page 255
- ":BUS<n>:LABel" on page 256
- ":BUS<n>:MASK" on page 257

:BUS<n>:DISPlay

(see page 1292)

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.

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" on page 250
- ":BUS<n>:BIT<m>" on page 251
- ":BUS<n>:BITS" on page 252
- ":BUS<n>:CLEar" on page 254
- ":BUS<n>:LABel" on page 256
- ":BUS<n>:MASK" on page 257

:BUS<n>:LABel

(see page 1292)

Command Syntax

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

<quoted_string> ::= any series of 32 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 Label strings are 32 characters or less, and may contain any commonly used ASCII characters. Labels with more than 32 characters are truncated to 32 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 32 or less characters as a
quoted ASCII string.
```

See Also

- "Introduction to :BUS<n> Commands" on page 250
- ":BUS<n>:BIT<m>" on page 251
- ":BUS<n>:BITS" on page 252
- ":BUS<n>:CLEar" on page 254
- ":BUS<n>:DISPlay" on page 255
- ":BUS<n>:MASK" on page 257
- ":CHANnel<n>:LABel" on page 278
- ":DISPlay:LABList" on page 351
- ":DIGital<d>:LABel" on page 315

Example Code

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

:BUS<n>:MASK

 (see page 1292)

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.

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" on page 250
- ":BUS<n>:BIT<m>" on page 251
- ":BUS<n>:BITS" on page 252
- ":BUS<n>:CLEar" on page 254
- ":BUS<n>:DISPlay" on page 255
- ":BUS<n>:LABel" on page 256
- "Commands Not Supported in Multiple Program Message Units" on page 1297

10 :CALibrate Commands

Utility commands for viewing calibration status and for starting the user calibration procedure. See "Introduction to :CALibrate Commands" on page 260.

Table 75 :CALibrate Commands Summary

Command	Query	Options and Query Returns
n/a	:CALibrate:DATE? (see page 261)	<return value> ::= <year>,<month>,<day>; all in NR1 format
:CALibrate:LABel <string> (see page 262)	:CALibrate:LABel? (see page 262)	<string> ::= quoted ASCII string up to 32 characters
:CALibrate:OUTPut <signal> (see page 263)	:CALibrate:OUTPut? (see page 263)	<signal> ::= {TRIGgers | MASK | WAVEgen | WGEN1 | WGEN2 | TSOurce} Note: WAVE and WGEN1 are equivalent. Note: WGEN2 only available on models with 2 WaveGen outputs.
n/a	:CALibrate:PROTected? (see page 264)	{"PROTected" | "UNPRotected"}
:CALibrate:STARt (see page 265)	n/a	n/a
n/a	:CALibrate:STATus? (see page 266)	<return value> ::= <status_code>,<status_string> <status_code> ::= an integer status code <status_string> ::= an ASCII status string

Table 75 :CALibrate Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:CALibrate:TEMPerature? (see page 267)	<return value> ::= degrees C delta since last cal in NR3 format
n/a	:CALibrate:TIME? (see page 268)	<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:DATE?

 (see page 1292)

Query Syntax `:CALibrate:DATE?`

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

Return Format `<date><NL>`

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

See Also

- "Introduction to :CALibrate Commands" on page 260

:CALibrate:LABel

(see page 1292)

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

See Also

- "Introduction to :CALibrate Commands" on page 260

:CALibrate:OUTPut

N (see page 1292)

Command Syntax

```
:CALibrate:OUTPut <signal>

<signal> ::= {TRIGgers | MASK | WAVEgen | WGEN1 | TSOurce}

Note: WAVE and WGEN1 are equivalent.
```

The CALibrate:OUTPut command sets the signal that is available on the rear panel AUX OUT BNC:

- TRIGgers – pulse when a trigger event occurs. The output level is 0-5 V into an open circuit, and 0-2.5 V into 50 Ω.
- MASK – signal from mask test indicating a failure.
- WAVEgen, WGEN1 – waveform generator sync output signal. This signal depends on the :WGEN<w>:FUNCtion setting:

Waveform Type	Sync Signal Characteristics
SINusoid, SQUare, RAMP, PULSe, SINC, EXPRise, EXPFall, GAUSsian	The Sync signal is a TTL positive pulse that occurs when the waveform rises above zero volts (or the DC offset value).
DC, NOISe, CARDiac	N/A

- TSOurce – The raw trigger signal from the oscilloscope's trigger circuit is output to Aux Out. It produces a rising edge whenever the input source would cause a trigger, even though that might occur multiple times within the time of a single acquisition. The trigger source can be a front panel analog input channel or an external trigger input. The output level is 0-5 V into an open circuit, and 0-2.5 V into 50 Ω. This option is not available in all trigger modes.

Query Syntax

```
:CALibrate:OUTPut?
```

The :CALibrate:OUTPut query returns the current source of the AUX OUT BNC signal.

Return Format

```
<signal><NL>

<signal> ::= {TRIG | MASK | WAVE | TSO}
```

See Also

- "Introduction to :CALibrate Commands" on page 260
- ":WGEN<w>:FUNCtion" on page 1197

:CALibrate:PROTected

(see page 1292)

Command Syntax

```
:CALibrate:PROTected {{0 | OFF} | {1 | ON}}
```

The :CALibrate:PROTected command disables or enables the Cal Protect control which allows user calibration to be run or prevents user calibration from being run.

Query Syntax

```
:CALibrate:PROTected?
```

The :CALibrate:PROTected? query returns the calibration protect (Cal Protect) control state. The value "PROTected" indicates user calibration will be prevented, and "UNPRotected" indicates calibration will be allowed.

Return Format

```
<control><NL>

<control> ::= {"PROTected" | "UNPRotected"}
```

See Also

- "Introduction to :CALibrate Commands" on page 260
- ":CALibrate:STARt" on page 265

:CALibrate:STARt

(see page 1292)

Command Syntax `:CALibrate:STARt`

The CALibrate:STARt command starts the user calibration procedure.

NOTE **Before starting the user calibration procedure, you must clear the Cal Protect control (see :CALibrate:PROTected). See the *User's Guide* for more details on user calibration.**

See Also

- "Introduction to :CALibrate Commands" on page 260
- ":CALibrate:PROTected" on page 264

:CALibrate:STATus?

(see page 1292)

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> ::= <status_code>,<status_string>

<status_code> ::= an integer status code

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

The status codes and strings can be:

Status Code	Status String
+0	Calibrated
-1	Not Calibrated

See Also

- "Introduction to :CALibrate Commands" on page 260

:CALibrate:TEMPerature?

N (see page 1292)

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" on page 260

:CALibrate:TIME?

(see page 1292)

Query Syntax `:CALibrate:TIME?`

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

Return Format `<time><NL>`

`<time> ::= hour,minute,second in NR1 format`

See Also

- "Introduction to :CALibrate Commands" on page 260

11 :CHANnel<n> Commands

Control all oscilloscope functions associated with individual analog channels or groups of channels. See "Introduction to :CHANnel<n> Commands" on page 272.

Table 76 :CHANnel<n> Commands Summary

Command	Query	Options and Query Returns
:CHANnel<n>:BWLimit {{0 | OFF} | {1 | ON}} (see page 273)	:CHANnel<n>:BWLimit? (see page 273)	{0 | 1} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:COUPling <coupling> (see page 274)	:CHANnel<n>:COUPling? (see page 274)	<coupling> ::= {AC | DC} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:DISPlay {{0 | OFF} | {1 | ON}} (see page 275)	:CHANnel<n>:DISPlay? (see page 275)	{0 | 1} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:IMPedance <impedance> (see page 276)	:CHANnel<n>:IMPedance? (see page 276)	<impedance> ::= {ONEMeg | FIFTy} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:INVert {{0 | OFF} | {1 | ON}} (see page 277)	:CHANnel<n>:INVert? (see page 277)	{0 | 1} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:LABel <string> (see page 278)	:CHANnel<n>:LABel? (see page 278)	<string> ::= any series of 32 or less ASCII characters enclosed in quotation marks <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:OFFSet <offset>[suffix] (see page 279)	:CHANnel<n>:OFFSet? (see page 279)	<offset> ::= Vertical offset value in NR3 format [suffix] ::= {V | mV} <n> ::= 1-2 or 1-4; in NR1 format

Table 76 :CHANnel<n> Commands Summary (continued)

Command	Query	Options and Query Returns
:CHANnel<n>:PROBe <attenuation> (see page 280)	:CHANnel<n>:PROBe? (see page 280)	<attenuation> ::= Probe attenuation ratio in NR3 format <n> ::= 1-2 or 1-4; in NR1 format
:CHANnel<n>:PROBe:BTN <setting> (see page 281)	:CHANnel<n>:PROBe:BTN? (see page 281)	<n> ::= 1 to (# analog channels) in NR1 format <setting> ::= {HEADlight | INFiniimode | RSTop | SINGle | CDISplay | AUToscale | FTRigger | QACTion | NACTion | NONE}
:CHANnel<n>:PROBe:CALibration (see page 282)	n/a	n/a
:CHANnel<n>:PROBe:EXTernal {{0 | OFF} | {1 | ON}} (see page 283)	:CHANnel<n>:PROBe:EXTernal? (see page 283)	<n> ::= 1 to (# analog channels) in NR1 format <setting> ::= {0 | 1}
:CHANnel<n>:PROBe:EXTernal:GAIN <gain_factor> (see page 284)	:CHANnel<n>:PROBe:EXTernal:GAIN? (see page 284)	<n> ::= 1 to (# analog channels) in NR1 format <gain_factor> ::= a real number from 0.0001 to 1000 in NR3 format
:CHANnel<n>:PROBe:EXTernal:UNITs <units> (see page 285)	:CHANnel<n>:PROBe:EXTernal:UNITs? (see page 285)	<n> ::= 1 to (# analog channels) in NR1 format <units> ::= {VOLT | AMPere}
:CHANnel<n>:PROBe:HEAD[:TYPE] <head_param> (see page 286)	:CHANnel<n>:PROBe:HEAD[:TYPE]? (see page 286)	<head_param> ::= {SEND0 | SEND6 | SEND12 | SEND20 | DIFF0 | DIFF6 | DIFF12 | DIFF20 | DSMA | DSMA6 | NONE} <n> ::= 1 to (# analog channels) in NR1 format
n/a	:CHANnel<n>:PROBe:ID? (see page 287)	<probe id> ::= unquoted ASCII string up to 11 characters <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:PROBe:MMODel <value> (see page 288)	:CHANnel<n>:PROBe:MMODel? (see page 288)	<value> ::= {P5205 | P5210 | P6205 | P6241 | P6243 | P6245 | P6246 | P6247 | P6248 | P6249 | P6250 | P6251 | P670X | P671X | TCP202} <n> ::= 1 to (# analog channels) in NR1 format

Table 76 :CHANnel<n> Commands Summary (continued)

Command	Query	Options and Query Returns
:CHANnel<n>:PROBe:MODE <setting> (see page 289)	:CHANnel<n>:PROBe:MODE? (see page 289)	<n> ::= 1 to (# analog channels) in NR1 format <setting> ::= {DIFFerential | DOFFset | SEA | SEB | CM}
:CHANnel<n>:PROBe:RSENse <value> (see page 290)	:CHANnel<n>:PROBe:RSENse? (see page 290)	<value> ::= Ohms in NR3 format <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:PROBe:SKEW <skew_value> (see page 291)	:CHANnel<n>:PROBe:SKEW? (see page 291)	<skew_value> ::= -100 ns to +100 ns in NR3 format <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:PROBe:STYPe <signal type> (see page 292)	:CHANnel<n>:PROBe:STYPe? (see page 292)	<signal type> ::= {DIFFerential | SINGle} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:PROBe:ZOOM {{0 | OFF} | {1 | ON}} (see page 293)	:CHANnel<n>:PROBe:ZOOM? (see page 293)	<setting> ::= {0 | 1} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:PROTection (see page 294)	:CHANnel<n>:PROTection? (see page 294)	{NORM | TRIP} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:RANGe <range>[suffix] (see page 295)	:CHANnel<n>:RANGe? (see page 295)	<range> ::= Vertical full-scale range value in NR3 format [suffix] ::= {V | mV} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:SCALe <scale>[suffix] (see page 296)	:CHANnel<n>:SCALe? (see page 296)	<scale> ::= Vertical units per division value in NR3 format [suffix] ::= {V | mV} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:UNITs <units> (see page 297)	:CHANnel<n>:UNITs? (see page 297)	<units> ::= {VOLT | AMPere} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:VERNier {{0 | OFF} | {1 | ON}} (see page 298)	:CHANnel<n>:VERNier? (see page 298)	{0 | 1} <n> ::= 1 to (# analog channels) in NR1 format

Introduction to :CHANnel<n> Commands

```
<n> ::= 1 to (# analog channels) in NR1 format
```

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 32 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 +0.0E+00;COUP DC;IMP ONEM;DISP 1;
BAND +5.00000000E+08;INV 0;LAB "1";UNIT VOLT;PROB +10.000000E+00;
PROB:SKEW +0.0E+00;STYP SING;EXT 0;EXT:GAIN +1.0000E+00
```

:CHANnel<n>:BWLimit

(see page 1292)

Command Syntax

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

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

<n> ::= 1 to (# analog channels) in NR1 format
```

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 40 MHz.

Note that there is also a global bandwidth limit (see :ACQuire:BANDwidth) that affects all analog input channels. When the channel bandwidth limit and the global bandwidth limit are both selected, the lower bandwidth limit is applied.

Query Syntax

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

The :CHANnel<n>:BWLimit? query returns the current setting of the channel bandwidth limit.

Return Format

```
<bwlimit><NL>

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

See Also

- ":ACQuire:BANDwidth" on page 232
- "Introduction to :CHANnel<n> Commands" on page 272

:CHANnel<n>:COUPling

(see page 1292)

Command Syntax

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

<coupling> ::= {AC | DC}

<n> ::= 1 to (# analog channels) in NR1 format
```

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" on page 272

:CHANnel<n>:DISPlay

(see page 1292)

Command Syntax

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

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

<n> ::= 1 to (# analog channels) in NR1 format
```

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" on page 272
- ":VIEW" on page 227
- ":BLANk" on page 217
- ":STATus?" on page 224
- ":POD<n>:DISPlay" on page 669
- ":DIGital<d>:DISPlay" on page 314

:CHANnel<n>:IMPedance

(see page 1292)

Command Syntax

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

<impedance> ::= {ONEMeg | FIFTy}

<n> ::= 1 to (# analog channels) in NR1 format
```

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).

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" on page 272

:CHANnel<n>:INVert

(see page 1292)

Command Syntax

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

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

<n> ::= 1 to (# analog channels) in NR1 format
```

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" on page 272

:CHANnel<n>:LABel

N (see page 1292)

Command Syntax

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

<string> ::= quoted ASCII string

<n> ::= 1 to (# analog channels) in NR1 format
```

NOTE Label strings are 32 characters or less, and may contain any commonly used ASCII characters. Labels with more than 32 characters are truncated to 32 characters.

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" on page 272
- ":DISPlay:LABel" on page 350
- ":DIGital<d>:LABel" on page 315
- ":DISPlay:LABList" on page 351
- ":BUS<n>:LABel" on page 256

Example Code

```
' LABEL - This command allows you to write a name (32 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 ch1 "CAL 1".
myScope.WriteString ":CHANnel2:LABel ""CAL2"""   ' Label ch1 "CAL2".
```

See complete example programs at: Chapter 44, “Programming Examples,” starting on page 1301

:CHANnel<n>:OFFSet

(see page 1292)

Command Syntax

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

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

<suffix> ::= {V | mV}

<n> ::= 1 to (# analog channels) in NR1 format
```

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 affected 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" on page 272
- ":CHANnel<n>:RANGe" on page 295
- ":CHANnel<n>:SCALe" on page 296
- ":CHANnel<n>:PROBe" on page 280

:CHANnel<n>:PROBe

(see page 1292)

Command Syntax

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

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

<n> ::= 1 to (# analog channels) in NR1 format
```

The :CHANnel<n>:PROBe command specifies the probe attenuation factor for the selected channel.

The probe attenuation factor may be from 0.001 to 10000.

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" on page 272
- ":CHANnel<n>:RANGe" on page 295
- ":CHANnel<n>:SCALe" on page 296
- ":CHANnel<n>:OFFSet" on page 279

Example Code

```
' CHANNEL_PROBE - Sets the probe attenuation factor for the
' selected channel.  The probe attenuation factor may be set
' from 0.001 to 10000.
myScope.WriteString ":CHANnel1:PROBe 10"   ' Set Probe to 10:1.
```

See complete example programs at: Chapter 44, “Programming Examples,” starting on page 1301

:CHANnel<n>:PROBe:BTN

N (see page 1292)

Command Syntax

```
:CHANnel<n>:PROBe:BTN <setting>

<setting> ::= {HEADlight | INFiniimode | RSTop | SINGle | CDISplay
   | AUToscale | FTRigger | QACTion | NACTion | NONE}

<n> ::= 1 to (# analog channels) in NR1 format
```

For N275xA InfiniiMode probes, the :CHANnel<n>:PROBe:BTN command sets the probe's quick-action button mode:

- HEADlight – Pressing the probe's quick-action button toggles the probe headlight on or off. Holding the quick-action button adjusts the intensity of the probe's headlight.
- INFiniimode – Cycles through the InfiniiMode settings (see :CHANnel<n>:PROBe:MODE).
- RSTop (Run/Stop) – Toggles between continuous data acquisition (running) and stopping data acquisitions (same as the oscilloscope's **[Run/Stop]** key).
- SINGle – Performs a single acquisition on the oscilloscope (same as the oscilloscope's **[Single]** key).
- CDISplay – Clears the oscilloscope display.
- AUToscale – Performs an oscilloscope Autoscale operation.
- FTRigger (Force Trigger) – Performs an oscilloscope "force trigger".
- QACTion (Quick Action) – Performs the quick action configured on the oscilloscope (see the oscilloscope's *user's guide* for more information).
- NACTion (No Action) – Disables the probe's quick-action button.
- NONE – When a probe does not have a quick-action button, this is the only valid setting.

Query Syntax

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

The :CHANnel<n>:PROBe:BTN? query returns the N275xA InfiniiMode probe's quick-action button setting.

Return Format

```
<setting><NL>

<setting> ::= {HEAD | INF | RST | SING | CDIS | AUT | FTR | QACT
   | NACT | NONE}
```

See Also

- ":CHANnel<n>:PROBe:MODE" on page 289

:CHANnel<n>:PROBe:CALibration

 (see page 1292)

Command Syntax `:CHANnel<n>:PROBe:CALibration`

The :CHANnel<n>:PROBe:CALibration command begins the probe degauss operation. For the N7026A and N2893A probes only.

:CHANnel<n>:PROBe:EXTernal

(see page 1292)

Command Syntax

```
:CHANnel<n>:PROBe:EXTernal {{0 | OFF} | {1 | ON}}

<n> ::= 1 to (# analog channels) in NR1 format
```

The :CHANnel<n>:PROBe:EXTernal command enables or disables External Scaling on the analog input channel. External Scaling can apply additional gain to the input channel to account for additional attenuators, adapters, etc., in the probing system.

Query Syntax

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

The :CHANnel<n>:PROBe:EXTernal? query returns whether External Scaling is enabled (1) or disabled (0).

Return Format

```
<setting><NL>

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

See Also

- ":CHANnel<n>:PROBe:EXTernal:GAIN" on page 284
- ":CHANnel<n>:PROBe:EXTernal:UNITs" on page 285

:CHANnel<n>:PROBe:EXTernal:GAIN

N (see page 1292)

Command Syntax

```
:CHANnel<n>:PROBe:EXTernal:GAIN <gain_factor>

<gain_factor> ::= a real number from 0.0001 to 1000 in NR3 format

<n> ::= 1 to (# analog channels) in NR1 format
```

The :CHANnel<n>:PROBe:EXTernal:GAIN command sets the gain associated with External Scaling.

Query Syntax

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

The :CHANnel<n>:PROBe:EXTernal:GAIN? query returns the External Scaling gain setting.

Return Format

```
<gain_factor><NL>
```

See Also

- ":CHANnel<n>:PROBe:EXTernal" on page 283
- ":CHANnel<n>:PROBe:EXTernal:UNITs" on page 285

:CHANnel<n>:PROBe:EXTernal:UNITs

N (see page 1292)

Command Syntax

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

<units> ::= {VOLT | AMPere}

<n> ::= 1 to (# analog channels) in NR1 format
```

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

Query Syntax

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

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

Return Format

```
<units><NL>

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

See Also

- ":CHANnel<n>:PROBe:EXTernal" on page 283
- ":CHANnel<n>:PROBe:EXTernal:GAIN" on page 284

:CHANnel<n>:PROBe:HEAD[:TYPE]

(see page 1292)

Command Syntax

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

```
:CHANnel<n>:PROBe:HEAD[:TYPE] <head_param>

<head_param> ::= {SEND0 | SEND6 | SEND12 | SEND20 | DIFF0 | DIFF6
                  | DIFF12 | DIFF20 | DSMA | DSMA6 | NONE}

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

The :CHANnel<n>:PROBe:HEAD[:TYPE] command sets an analog channel probe head type and dB value. You can choose from:

- SEND0 – Single-ended, 0 dB.
- SEND6 – Single-ended, 6 dB.
- SEND12 – Single-ended, 12 dB.
- SEND20 – Single-ended, 20 dB.
- DIFF0 – Differential, 0 dB.
- DIFF6 – Differential, 6 dB.
- DIFF12 – Differential, 12 dB.
- DIFF20 – Differential, 20 dB.
- DSMA – Differential SMA probe head, 0 dB.
- DSMA6 – Differential SMA probe head, 6 dB.

Query Syntax

```
:CHANnel<n>:PROBe:HEAD[:TYPE]?
```

The :CHANnel<n>:PROBe:HEAD[:TYPE]? query returns the current probe head type setting for the selected channel.

Return Format

```
<head_param><NL>

<head_param> ::= {SEND0 | SEND6 | SEND12 | SEND20 | DIFF0 | DIFF6
                  | DIFF12 | DIFF20 | DSMA | DSMA6 | NONE}
```

See Also

- "Introduction to :CHANnel<n> Commands" on page 272
- ":CHANnel<n>:PROBe" on page 280
- ":CHANnel<n>:PROBe:ID?" on page 287
- ":CHANnel<n>:PROBe:SKEW" on page 291
- ":CHANnel<n>:PROBe:STYPe" on page 292

:CHANnel<n>:PROBe:ID?

(see page 1292)

Query Syntax

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

<n> ::= 1 to (# analog channels) in NR1 format
```

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" on page 272
- "Queries Not Supported in Multiple Program Message Units" on page 1297

:CHANnel<n>:PROBe:MMODel

N (see page 1292)

Command Syntax

```
:CHANnel<n>:PROBe:MMODel <value>

<value> ::= {P5205 | P5210 | P6205 | P6241 | P6243 | P6245 | P6246
   | P6247 | P6248 | P6249 | P6250 | P6251 | P670X | P671X | TCP202}

<n> ::= 1 to (# analog channels) in NR1 format
```

The :CHANnel<n>:PROBe:MMODel command sets the model number of the supported Tektronix probe.

Query Syntax

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

The :CHANnel<n>:PROBe:MMODel? query returns the model number setting.

Return Format

```
<value><NL>

<value> ::= {P5205 | P5210 | P6205 | P6241 | P6243 | P6245 | P6246
   | P6247 | P6248 | P6249 | P6250 | P6251 | P670X | P671X | TCP202}
```

See Also

- ":CHANnel<n>:PROBe:ID?" on page 287

:CHANnel<n>:PROBe:MODE

N (see page 1292)

Command Syntax

```
:CHANnel<n>:PROBe:MODE <setting>

<setting> ::= {DIFFerential | DOFFset | SEA | SEB | CM}

<n> ::= 1 to (# analog channels) in NR1 format
```

For N275xA InfiniiMode probes, the :CHANnel<n>:PROBe:MODE command sets the probe's mode:

- DIFFerential – Differential.
- DOFFset – Differential with Offset.
- SEA – Single-Ended, side A.
- SEB – Single-Ended, side B.
- CM – Common Mode.

Query Syntax

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

The :CHANnel<n>:PROBe:MODE? query returns the N275xA InfiniiMode probe's mode setting.

Return Format

```
<setting><NL>

<setting> ::= {DIFF | DOFF | SEA | SEB | CM}
```

See Also

- ":CHANnel<n>:PROBe:BTN" on page 281

:CHANnel<n>:PROBe:RSENse

(see page 1292)

Command Syntax

```
:CHANnel<n>:PROBe:RSENse <value>

<value> ::= Ohms in NR3 format

<n> ::= 1 to (# analog channels) in NR1 format
```

When the N2820A high-sensitivity current probe is used with the N2825A user-defined R-sense head, the :CHANnel<n>:PROBe:RSENse command specifies the value of the R-sense resistor that is being probed in the device under test (DUT). Suppoeted R-sense resistor values are from 1 mΩ to 1MΩ.

Query Syntax

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

The :CHANnel<n>:PROBe:RSENse? query returns the R-sense resistor value setting.

When the N2820A high-sensitivity current probe is not attached, the query returns "INF".

When the N2820A high-sensitivity current probe is attached and a special head is attached to the probe, the query returns the value of the special head. For example, if the head is a "20 mΩ" head, the query returns 0.02.

Return Format

```
{<value> | INF}<NL>

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

See Also

- ":CHANnel<n>:PROBe:ZOOM" on page 293

:CHANnel<n>:PROBe:SKEW

(see page 1292)

Command Syntax

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

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

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

<n> ::= 1 to (# analog channels) in NR1 format
```

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" on page 272

:CHANnel<n>:PROBe:STYPe

(see page 1292)

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 to (# analog channels) in NR1 format
```

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" on page 272
- ":CHANnel<n>:OFFSet" on page 279

:CHANnel<n>:PROBe:ZOOM

 (see page 1292)

Command Syntax

```
:CHANnel<n>:PROBe:ZOOM {{0 | OFF} | {1 | ON}}

<n> ::= 1 to (# analog channels) in NR1 format
```

When the N2820A high-sensitivity current probe is used with both the Primary and Secondary cables, the :CHANnel<n>:PROBe:ZOOM command specifies whether this cable will have the Zoom In waveform (ON) or the Zoom Out waveform (OFF). The other cable will have the opposite waveform.

Query Syntax

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

The :CHANnel<n>:PROBe:ZOOM? query returns the zoom setting.

Return Format

```
<setting><NL>

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

See Also

- ":CHANnel<n>:PROBe:RSENse" on page 290

:CHANnel<n>:PROTection

(see page 1292)

Command Syntax

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

<n> ::= 1 to (# analog channels) in NR1 format| 4}
```

When the analog channel input impedance is set to 50Ω, 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" on page 276) 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" on page 272
- ":CHANnel<n>:COUPling" on page 274
- ":CHANnel<n>:IMPedance" on page 276
- ":CHANnel<n>:PROBe" on page 280

:CHANnel<n>:RANGe

(see page 1292)

Command Syntax

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

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

<suffix> ::= {V | mV}

<n> ::= 1 to (# analog channels) in NR1 format
```

The :CHANnel<n>:RANGe command defines the full-scale vertical axis of the selected channel. When using 1:1 probe attenuation, legal values for the range are from 8 mV to 40 V.

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" on page 272
- ":CHANnel<n>:SCALe" on page 296
- ":CHANnel<n>:PROBe" on page 280

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.
```

See complete example programs at: Chapter 44, “Programming Examples,” starting on page 1301

:CHANnel<n>:SCALe

(see page 1292)

Command Syntax

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

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

<suffix> ::= {V | mV}

<n> ::= 1 to (# analog channels) in NR1 format
```

The :CHANnel<n>:SCALe command sets the vertical scale, or units per division, of the selected channel.

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" on page 272
- ":CHANnel<n>:RANGe" on page 295
- ":CHANnel<n>:PROBe" on page 280

:CHANnel<n>:UNITs

N (see page 1292)

Command Syntax

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

<units> ::= {VOLT | AMPere}

<n> ::= 1 to (# analog channels) in NR1 format
```

The :CHANnel<n>:UNITs command sets the measurement units for the connected probe. Select VOLT for a voltage probe and select AMPere 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" on page 272
- ":CHANnel<n>:RANGe" on page 295
- ":CHANnel<n>:PROBe" on page 280
- ":EXTernal:UNITs" on page 373

:CHANnel<n>:VERNier

(see page 1292)

Command Syntax

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

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

<n> ::= 1 to (# analog channels) in NR1 format
```

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" on page 272

12 :COUNter<c> Commands

These commands control the counter feature. See "Introduction to :COUNter<c> Commands" on page 299.

Table 77 :COUNter<c> Commands Summary

Command	Query	Options and Query Returns
n/a	:COUNter<c>:CURRent? (see page 301)	<c> ::= {1 | 2} <value> ::= current counter value in NR3 format
:COUNter<c>:ENABle {{0 | OFF} | {1 | ON}} (see page 302)	:COUNter<c>:ENABle? (see page 302)	<c> ::= {1 | 2} {0 | 1}
:COUNter<c>:MODE <mode> (see page 303)	:COUNter<c>:MODE (see page 303)	<c> ::= {1 | 2} <mode> ::= {FREQuency | PERiod | TOTalize}
:COUNter<c>:NDIGits <value> (see page 304)	:COUNter<c>:NDIGits (see page 304)	<c> ::= {1 | 2} <value> ::= 3 to 8 in NR1 format
:COUNter<c>:SOURce <source> (see page 305)	:COUNter<c>:SOURce? (see page 305)	<c> ::= {1 | 2} <source> ::= {CHANnel<n> | TQEVent} <n> ::= 1 to (# analog channels) in NR1 format
:COUNter<c>:TOTalize:CLEar (see page 306)	n/a	<c> ::= {1 | 2}
:COUNter<c>:TOTalize:SLOPe <slope> (see page 307)	:COUNter<c>:TOTalize:SLOPe? (see page 307)	<c> ::= {1 | 2} <slope> ::= {{NEGative | FALLing} | {POSitive | RISing}}

Introduction to :COUNter<c> Commands

The :COUNter subsystem provides commands to control the counter feature.

Reporting the Setup

Use :COUNter<c>? to query setup information for the COUNter subsystem.

Return Format

The following is a sample response from the :COUNter1? query. In this case, the query was issued following the *RST command.

```
:COUN1:ENAB 0;SOUR CHAN1;MODE FREQ;NDIG 5
```

:COUNter<c>:CURRent?

N (see page 1292)

Query Syntax

```
:COUNter<c>:CURRent?

<c> ::= {1 | 2}
```

The :COUNter<c>:CURRent? query returns the current counter value.

Return Format

```
<value><NL>

<value> ::= current counter value in NR3 format
```

See Also

- ":COUNter<c>:ENABle" on page 302
- ":COUNter<c>:MODE" on page 303
- ":COUNter<c>:NDIGits" on page 304
- ":COUNter<c>:SOURce" on page 305

:COUNter<c>:ENABle

 (see page 1292)

Command Syntax

```
:COUNter<c>:ENABle {{0 | OFF} | {1 | ON}}

<c> ::= {1 | 2}
```

The :COUNter<c>:ENABle command enables or disables the counter feature.

Query Syntax

```
:COUNter<c>:ENABle?
```

The :COUNter<c>:ENABle? query returns whether the counter is enabled or disabled.

Return Format

```
<off_on><NL>

{0 | 1}
```

See Also

- ":COUNter<c>:CURRent?" on page 301
- ":COUNter<c>:MODE" on page 303
- ":COUNter<c>:NDIGits" on page 304
- ":COUNter<c>:SOURce" on page 305

:COUNter<c>:MODE

N (see page 1292)

Command Syntax

```
:COUNter<c>:MODE <mode>

<c> ::= {1 | 2}

<mode> ::= {FREQuency | PERiod | TOTalize}
```

The :COUNter<c>:MODE command sets the counter mode:

- FREQuency — the cycles per second (Hz) of the signal.
- PERiod — the time periods of the signal's cycles.
- TOTalize — the count of edge events on the signal.

Query Syntax

```
:COUNter<c>:MODE?
```

The :COUNter<c>:MODE? query returns the counter mode setting.

Return Format

```
<mode><NL>

<mode> ::= {FREQ | PER | TOT}
```

See Also

- ":COUNter<c>:CURRent?" on page 301
- ":COUNter<c>:ENABle" on page 302
- ":COUNter<c>:NDIGits" on page 304
- ":COUNter<c>:SOURce" on page 305
- ":COUNter<c>:TOTalize:CLEar" on page 306
- ":COUNter<c>:TOTalize:SLOPe" on page 307

:COUNter<c>:NDIGits

(see page 1292)

Command Syntax

```
:COUNter<c>:NDIGits <value>

<c> ::= {1 | 2}

<value> ::= 3 to 8 in NR1 format
```

The :COUNter<c>:NDIGits command sets the number of digits of resolution used for the frequency or period counter.

Higher resolutions require longer gate times, which cause the measurement times to be longer as well.

Query Syntax

```
:COUNter<c>:NDIGits?
```

The :COUNter<c>:NDIGits? query returns the currently set number of digits of resolution.

Return Format

```
<value><NL>

<value> ::= 3 to 8 in NR1 format
```

See Also

- ":COUNter<c>:CURRent?" on page 301
- ":COUNter<c>:ENABle" on page 302
- ":COUNter<c>:MODE" on page 303
- ":COUNter<c>:SOURce" on page 305
- ":COUNter<c>:TOTalize:CLEar" on page 306
- ":COUNter<c>:TOTalize:SLOPe" on page 307

:COUNter<c>:SOURce

N (see page 1292)

Command Syntax

```
:COUNter<c>:SOURce <source>

<c> ::= {1 | 2}

<source> ::= {CHANnel<n> | TQEVent}

<n> ::= 1 to (# analog channels) in NR1 format
```

The :COUNter<c>:SOURce command selects the waveform source that the counter measures. You can select one of the analog input channels or the trigger qualified event signal.

With the TQEVent (trigger qualified event) source (available when the trigger mode is not EDGE), you can see how often trigger events are detected. This can be more often than when triggers actually occur, due to the oscilloscope's acquisition time or update rate capabilities. The AUX OUT signal shows when triggers actually occur. Remember that the oscilloscope's trigger circuitry does not re-arm until the holdoff time occurs (:TRIGger:HOLDoff) and that the minimum holdoff time is 40 ns; therefore, the maximum trigger qualified event frequency that can be counted is 25 MHz.

Query Syntax

```
:COUNter<c>:SOURce?
```

The :COUNter<c>:SOURce? query returns the currently set counter source channel.

Return Format

```
<source><NL>
```

See Also

- ":COUNter<c>:CURRent?" on page 301
- ":COUNter<c>:ENABle" on page 302
- ":COUNter<c>:MODE" on page 303
- ":COUNter<c>:NDIGits" on page 304

:COUNter<c>:TOTalize:CLEar

N (see page 1292)

Command Syntax

```
:COUNter<c>:TOTalize:CLEar
<c> ::= {1 | 2}
```

The :COUNter<c>:TOTalize:CLEar command zeros the edge event counter.

See Also

- ":COUNter<c>:CURRent?" on page 301
- ":COUNter<c>:ENABle" on page 302
- ":COUNter<c>:MODE" on page 303
- ":COUNter<c>:NDIGits" on page 304
- ":COUNter<c>:SOURce" on page 305
- ":COUNter<c>:TOTalize:SLOPe" on page 307

:COUNter<c>:TOTalize:SLOPe

(see page 1292)

Command Syntax

```
:COUNter<c>:TOTalize:SLOPe <slope>

<c> ::= {1 | 2}

<slope> ::= {{NEGative | FALLing} | {POSitive | RISing}}
```

The :COUNter<c>:TOTalize:SLOPe command specifies whether positive or negative edges are counted.

Query Syntax

```
:COUNter<c>:TOTalize:SLOPe?
```

The :COUNter<c>:TOTalize:SLOPe? query returns the currently set slope specification.

Return Format

```
<slope><NL>

<slope> ::= {NEG | POS}
```

See Also

- ":COUNter<c>:CURRent?" on page 301
- ":COUNter<c>:ENABle" on page 302
- ":COUNter<c>:MODE" on page 303
- ":COUNter<c>:NDIGits" on page 304
- ":COUNter<c>:SOURce" on page 305
- ":COUNter<c>:TOTalize:CLEar" on page 306

13 :DIGital<d> Commands

Control all oscilloscope functions associated with individual digital channels. See "Introduction to :DIGital<d> Commands" on page 310.

Table 78 :DIGital<d> Commands Summary

Command	Query	Options and Query Returns
:DIGital:ORDer:ASCending (see page 311)	n/a	n/a
:DIGital:ORDer:DESCending (see page 312)	n/a	n/a
:DIGital:ORDer[:SET] <digital_channels_list> (see page 313)	:DIGital:ORDer[:SET]? (see page 313)	<digital_channels_list> ::= comma-separated list, e.g., DIGital<d>,DIGital<d>,.. <d> ::= 0 to (# digital channels - 1) in NR1 format
:DIGital<d>:DISPlay {{0 | OFF} | {1 | ON}} (see page 314)	:DIGital<d>:DISPlay? (see page 314)	<d> ::= 0 to (# digital channels - 1) in NR1 format {0 | 1}
:DIGital<d>:LABel <string> (see page 315)	:DIGital<d>:LABel? (see page 315)	<d> ::= 0 to (# digital channels - 1) in NR1 format <string> ::= any series of 32 or less ASCII characters enclosed in quotation marks
n/a	:DIGital<d>:SETup? (see page 316)	<d> ::= 0 to (# digital channels - 1) in NR1 format Returns setup information string, for example, ":DIG0:DISP 1;THR +1.40E+00"

Table 78 :DIGital<d> Commands Summary (continued)

Command	Query	Options and Query Returns
:DIGital<d>:SIZE <value> (see page 317)	:DIGital<d>:SIZE? (see page 317)	<d> ::= 0 to (# digital channels - 1) in NR1 format <value> ::= {SMALl | MEDium | LARGe}
:DIGital<d>:THReshold <value>[suffix] (see page 318)	:DIGital<d>:THReshold? (see page 318)	<d> ::= 0 to (# digital channels - 1) 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<d> Commands

```
<d> ::= 0 to (# digital channels - 1) in NR1 format
```

The DIGital subsystem commands control the viewing, labeling, and positioning of digital channels. They also control threshold settings for groups of digital channels, or *pods*.

Reporting the Setup

Use :DIGital<d>:SETup? to query setup information for the DIGital subsystem.

Return Format

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

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

:DIGital:ORDer:ASCending

 (see page 1292)

Command Syntax `:DIGital:ORDer:ASCending`

The :DIGital:ORDer:ASCending command displays the digital channels that are turned on (see :DIGital<d>:DISPlay) in ascending order.

See Also

- ":DIGital<d>:DISPlay" on page 314
- ":DIGital:ORDer:DESCending" on page 312
- ":DIGital:ORDer[:SET]" on page 313

:DIGital:ORDer:DESCending

 (see page 1292)

Command Syntax `:DIGital:ORDer:DESCending`

The :DIGital:ORDer:DESCending command displays the digital channels that are turned on (see :DIGital<d>:DISPlay) in descending order.

See Also

- ":DIGital<d>:DISPlay" on page 314
- ":DIGital:ORDer:ASCending" on page 311
- ":DIGital:ORDer[:SET]" on page 313

:DIGital:ORDer[:SET]

N (see page 1292)

Command Syntax

```
:DIGital:ORDer[:SET] <digital_channels_list>

<digital_channels_list> ::= comma-separated list,
   e.g., DIGital<d>,DIGital<d>,..

<d> ::= 0 to (# digital channels - 1) in NR1 format
```

The :DIGital:ORDer[:SET] command displays digital channels in the order listed.

Digital channels not in the list are turned off and are not displayed.

Query Syntax

```
:DIGital:ORDer[:SET]?
```

The :DIGital:ORDer[:SET]? query returns the current order of displayed digital channels.

Return Format

```
<digital_channels_list><NL>

<digital_channels_list> ::= comma-separated list
```

See Also

- ":DIGital:ORDer:ASCending" on page 311
- ":DIGital:ORDer:DESCending" on page 312
- ":DIGital<d>:DISPlay" on page 314

:DIGital<d>:DISPlay

N (see page 1292)

Command Syntax

```
:DIGital<d>:DISPlay <display>

<d> ::= 0 to (# digital channels - 1) in NR1 format

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

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

Query Syntax

```
:DIGital<d>:DISPlay?
```

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

Return Format

```
<display><NL>

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

See Also

- "Introduction to :DIGital<d> Commands" on page 310
- ":DIGital<d>:SETup?" on page 316
- ":POD<n>:DISPlay" on page 669
- ":CHANnel<n>:DISPlay" on page 275
- ":VIEW" on page 227
- ":BLANk" on page 217
- ":STATus?" on page 224

:DIGital<d>:LABel

N (see page 1292)

Command Syntax

```
:DIGital<d>:LABel <string>

<d> ::= 0 to (# digital channels - 1) in NR1 format

<string> ::= any series of 32 or less characters as quoted ASCII string.
```

The :DIGital<d>: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 Label strings are 32 characters or less, and may contain any commonly used ASCII characters. Labels with more than 32 characters are truncated to 32 characters.

Query Syntax

```
:DIGital<d>:LABel?
```

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

Return Format

```
<label string><NL>

<label string> ::= any series of 32 or less characters as a quoted
                   ASCII string.
```

See Also

- "Introduction to :DIGital<d> Commands" on page 310
- ":CHANnel<n>:LABel" on page 278
- ":DISPlay:LABList" on page 351
- ":BUS<n>:LABel" on page 256

:DIGital<d>:SETup?

(see page 1292)

Query Syntax `:DIGital<d>:SETup?`

The :DIGital<d>:SETup? query returns setup information for the specified digital channel.

Return Format `<string><NL>`

`<string> ::= setup information string`

For example:

```
:DIG0:DISP 1;THR +1.40E+00
```

See Also

- ":DIGital<d>:DISPlay" on page 314
- ":DIGital<d>:THReshold" on page 318

:DIGital<d>:SIZE

N (see page 1292)

Command Syntax

```
:DIGital<d>:SIZE <value>

<d> ::= 0 to (# digital channels - 1) in NR1 format

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

The :DIGital<d>: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 all other as well.

Query Syntax

```
:DIGital<d>:SIZE?
```

The :DIGital<d>: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<d> Commands" on page 310
- ":POD<n>:SIZE" on page 672
- ":DIGital:ORDer[:SET]" on page 313

:DIGital<d>:THReshold

(see page 1292)

Command Syntax

```
:DIGital<d>:THReshold <value>

<d> ::= 0 to (# digital channels - 1) in NR1 format

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

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

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

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

The :DIGital<d>:THReshold command sets the logic threshold value for all channels in the same *pod* as the specified channel. The threshold is used for triggering purposes and for displaying the digital data as high (above the threshold) or low (below the threshold).

Query Syntax

```
:DIGital<d>:THReshold?
```

The :DIGital<d>: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<d> Commands" on page 310
- ":DIGital<d>:SETup?" on page 316
- ":POD<n>:THReshold" on page 673
- ":TRIGger[:EDGE]:LEVel" on page 1096

14 :DISPlay Commands

Control how waveforms, graticule, and text are displayed and written on the screen. See "Introduction to :DISPlay Commands" on page 322.

Table 79 :DISPlay Commands Summary

Command	Query	Options and Query Returns
:DISPlay:ANNotation<n> {{0 | OFF} | {1 | ON}} (see page 324)	:DISPlay:ANNotation<n>? (see page 324)	{0 | 1} <n> ::= an integer from 1 to 20 in NR1 format.
:DISPlay:ANNotation<n>:BACKground <mode> (see page 325)	:DISPlay:ANNotation<n>:BACKground? (see page 325)	<mode> ::= {OPAQue | INVerted | TRANsparent} <n> ::= an integer from 1 to 20 in NR1 format.
:DISPlay:ANNotation<n>:COLor <color> (see page 326)	:DISPlay:ANNotation<n>:COLor? (see page 326)	<color> ::= {CH1 | CH2 | CH3 | CH4 | DIG | MATH | REF | MARKer | WHITe | RED} <n> ::= an integer from 1 to 20 in NR1 format.
:DISPlay:ANNotation<n>:GRID <number> (see page 327)	:DISPlay:ANNotation<n>:GRID? (see page 327)	<number> ::= an integer from 1 to 20 in NR1 format. <n> ::= an integer from 1 to 20 in NR1 format.
:DISPlay:ANNotation<n>:MODE <mode> (see page 328)	:DISPlay:ANNotation<n>:MODE? (see page 328)	<mode> ::= {GRID | SOURce} <n> ::= an integer from 1 to 20 in NR1 format.
:DISPlay:ANNotation<n>:SOURce <source> (see page 329)	:DISPlay:ANNotation<n>:SOURce? (see page 329)	<source> ::= {CHANnel<n> | WMEMory<n> | FUNCtion<n> | FFT | DIGital<n>} <n> ::= an integer in NR1 format.
:DISPlay:ANNotation<n>:TEXT <string> (see page 330)	:DISPlay:ANNotation<n>:TEXT? (see page 330)	<string> ::= quoted ASCII string (up to 254 characters) <n> ::= an integer from 1 to 20 in NR1 format.

Table 79 :DISPlay Commands Summary (continued)

Command	Query	Options and Query Returns
:DISPlay:ANNotation<n>:XPOSition <value> (see page 331)	:DISPlay:ANNotation<n>:XPOSition? (see page 331)	<value> ::= a decimal from 0.0 to 1.0 in NR3 format that is a percent of the grid width. <n> ::= an integer from 1 to 20 in NR1 format.
:DISPlay:ANNotation<n>:YPOSition <value> (see page 332)	:DISPlay:ANNotation<n>:YPOSition? (see page 332)	<value> ::= a decimal from 0.0 to 1.0 in NR3 format that is a percent of the grid height. <n> ::= an integer from 1 to 20 in NR1 format.
:DISPlay:BACKlight {{0 | OFF} | {1 | ON}} (see page 333)	n/a	n/a
:DISPlay:CLEar (see page 334)	n/a	n/a
:DISPlay:CLOCk:IGRid {{0 | OFF} | {1 | ON}} (see page 335)	:DISPlay:CLOCk:IGRid? (see page 335)	<setting> ::= {0 | 1}
:DISPlay:CLOCk:IGRid:XPOSition <value> (see page 336)	:DISPlay:CLOCk:IGRid:XPOSition? (see page 336)	<value> ::= a decimal from 0.0 to 1.0 in NR3 format that is a percent of the grid width.
:DISPlay:CLOCk:IGRid:YPOSition <value> (see page 337)	:DISPlay:CLOCk:IGRid:YPOSition? (see page 337)	<value> ::= a decimal from 0.0 to 1.0 in NR3 format that is a percent of the grid height.
:DISPlay:CLOCk[:STATe] {{0 | OFF} | {1 | ON}} (see page 338)	:DISPlay:CLOCk[:STATe]? (see page 338)	<setting> ::= {0 | 1}
n/a	:DISPlay:DATA? [<format>] (see page 339)	<format> ::= {BMP | PNG} <display data> ::= data in IEEE 488.2 # format
:DISPlay:GRATicule:ALABels {{0 | OFF} | {1 | ON}} (see page 340)	:DISPlay:GRATicule:ALABels? (see page 340)	<setting> ::= {0 | 1}
:DISPlay:GRATicule:ALABels:DUAL {{0 | OFF} | {1 | ON}} (see page 341)	:DISPlay:GRATicule:ALABels:DUAL? (see page 341)	<setting> ::= {0 | 1}

Table 79 :DISPlay Commands Summary (continued)

Command	Query	Options and Query Returns
:DISPlay:GRATicule:ALABels:IGRid {{0 | OFF} | {1 | ON}} (see page 342)	:DISPlay:GRATicule:ALABels:IGRid? (see page 342)	<setting> ::= {0 | 1}
:DISPlay:GRATicule:COUNt <value> (see page 343)	:DISPlay:GRATicule:COUNt? (see page 343)	<num_grids> ::= an integer from 1 to 4 in NR1 format.
:DISPlay:GRATicule:FSCReen {{0 | OFF} | {1 | ON}} (see page 344)	:DISPlay:GRATicule:FSCReen? (see page 344)	<setting> ::= {0 | 1}
:DISPlay:GRATicule:INTensity <value> (see page 345)	:DISPlay:GRATicule:INTensity? (see page 345)	<value> ::= an integer from 0 to 100 in NR1 format.
:DISPlay:GRATicule:LAYout <type> (see page 346)	:DISPlay:GRATicule:LAYout? (see page 346)	<type> ::= {STACked | TILed | OVERlay | CUSTom}
:DISPlay:GRATicule:SET <source>,<grid> (see page 347)	n/a	<source> ::= {CHANnel<n> | WMEMory<n> | FUNCtion<n> | FFT | SBUS<n> | GAIN | PHASe} <grid> ::= a grid number integer from 1 to 4 in NR1 format.
n/a	:DISPlay:GRATicule:SOURce? <grid> (see page 348)	<grid> ::= a grid number integer from 1 to 4 in NR1 format. Returns: <sources> ::= comma-separated list of the waveforms displayed in the grid. Can contain items like: CHAN<n>, WMEM<n>, FUNC<n>, FFT, SBUS<n>, GAIN, PHASe
:DISPlay:INTensity:WAVeform <value> (see page 349)	:DISPlay:INTensity:WAVeform? (see page 349)	<value> ::= an integer from 0 to 100 in NR1 format.
:DISPlay:LABel {{0 | OFF} | {1 | ON}} (see page 350)	:DISPlay:LABel? (see page 350)	{0 | 1}
:DISPlay:LABList <binary block> (see page 351)	:DISPlay:LABList? (see page 351)	<binary block> ::= an ordered list of up to 75 labels, each 10 characters maximum, separated by newline characters
:DISPlay:MESSage:CLEar (see page 352)	n/a	n/a

Table 79 :DISPlay Commands Summary (continued)

Command	Query	Options and Query Returns
:DISPlay:PERSistence <value> (see page 353)	:DISPlay:PERSistence? (see page 353)	<value> ::= {MINimum | INFinite | <time>} <time> ::= seconds in in NR3 format from 100E-3 to 60E0
:DISPlay:PERSistence:CLEar (see page 354)	n/a	n/a
n/a	:DISPlay:RESults:CATalog? (see page 355)	<windows> ::= list of the results windows displayed from left to right. Can contain: MEAS, MARK, HIST, DVM, COUN, LIST, MTES, FRAN
:DISPlay:RESults:LAYout <format> (see page 356)	:DISPlay:RESults:LAYout? (see page 356)	<format> ::= {TAB | LIST}
:DISPlay:RESults:LAYout:LIST[:SELect] <results_window> (see page 357)	:DISPlay:RESults:LAYout:LIST[:SELect]? (see page 357)	<results_window> ::= {MEAS | MARK | HIST | DVM | COUN | LIST | MTES | FRAN}
:DISPlay:RESults:LAYout:TAB:LEFT[:SELect] <results_window> (see page 358)	:DISPlay:RESults:LAYout:TAB:LEFT[:SELect]? (see page 358)	<results_window> ::= {MEAS | MARK | HIST | DVM | COUN | LIST | MTES | FRAN}
:DISPlay:RESults:LAYout:TAB:RIGHt[:SELect] <results_window> (see page 359)	:DISPlay:RESults:LAYout:TAB:RIGHt[:SELect]? (see page 359)	<results_window> ::= {MEAS | MARK | HIST | DVM | COUN | LIST | MTES | FRAN}
:DISPlay:RESults:SIZE <size> (see page 360)	:DISPlay:RESults:SIZE? (see page 360)	<size> ::= {CUSTom,<height> | FULL | HIDDen} <height> ::= a pixel height integer from 36 to 1024 in NR1 format.
:DISPlay:TRANsparent {OFF | ON} (see page 361)	:DISPlay:TRANsparent? (see page 361)	{OFF | ON}

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.
- 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;PERS MIN
```

:DISPlay:ANNotation<n>

N (see page 1292)

Command Syntax

```
:DISPlay:ANNotation<n> <setting>

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

<n> ::= an integer from 1 to 20 in NR1 format.
```

The :DISPlay:ANNotation<n> command turns the annotation on and off. When on, the annotation appears in the upper left corner of the oscilloscope's display.

The annotation is useful for documentation purposes, to add notes before capturing screens.

Query Syntax

```
:DISPlay:ANNotation<n>?
```

The :DISPlay:ANNotation<n>? query returns the annotation setting.

Return Format

```
<value><NL>

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

See Also

- ":DISPlay:ANNotation<n>:TEXT" on page 330
- ":DISPlay:ANNotation<n>:COLor" on page 326
- ":DISPlay:ANNotation<n>:BACKground" on page 325
- ":DISPlay:ANNotation<n>:XPOSition" on page 331
- ":DISPlay:ANNotation<n>:YPOSition" on page 332
- "Introduction to :DISPlay Commands" on page 322

:DISPlay:ANNotation<n>:BACKground

N (see page 1292)

Command Syntax

```
:DISPlay:ANNotation<n>:BACKground <mode>

<mode> ::= {OPAQue | INVerted | TRANsparent}

<n> ::= an integer from 1 to 20 in NR1 format.
```

The :DISPlay:ANNotation<n>:BACKground command specifies the background of the annotation:

- OPAQue – the annotation has a solid background.
- INVerted – the annotation's foreground and background colors are switched.
- TRANsparent – the annotation has a transparent background.

Query Syntax

```
:DISPlay:ANNotation<n>:BACKground?
```

The :DISPlay:ANNotation<n>:BACKground? query returns the specified annotation background mode.

Return Format

```
<mode><NL>

<mode> ::= {OPAQ | INV | TRAN}
```

See Also

- ":DISPlay:ANNotation<n>" on page 324
- ":DISPlay:ANNotation<n>:TEXT" on page 330
- ":DISPlay:ANNotation<n>:COLor" on page 326
- ":DISPlay:ANNotation<n>:XPOSition" on page 331
- ":DISPlay:ANNotation<n>:YPOSition" on page 332
- "Introduction to :DISPlay Commands" on page 322

:DISPlay:ANNotation<n>:COLor

(see page 1292)

Command Syntax

```
:DISPlay:ANNotation<n>:COLor <color>

<color> ::= {CH1 | CH2 | CH3 | CH4 | DIG | MATH | REF | MARKer | WHITe
             | RED}

<n> ::= an integer from 1 to 20 in NR1 format.
```

When an annotation's mode is GRID (see :DISPlay:ANNotation<n>:MODE), the :DISPlay:ANNotation<n>:COLor command specifies the annotation color. You can choose white, red, or colors that match analog channels, digital channels, math waveforms, reference waveforms, or markers.

Query Syntax

```
:DISPlay:ANNotation<n>:COLor?
```

The :DISPlay:ANNotation<n>:COLor? query returns the specified annotation color.

Return Format

```
<color><NL>

<color> ::= {CH1 | CH2 | CH3 | CH4 | DIG | MATH | REF | MARK | WHIT
             | RED}
```

See Also

- ":DISPlay:ANNotation<n>:MODE" on page 328
- ":DISPlay:ANNotation<n>:GRID" on page 327
- ":DISPlay:ANNotation<n>" on page 324
- ":DISPlay:ANNotation<n>:TEXT" on page 330
- ":DISPlay:ANNotation<n>:BACKground" on page 325
- ":DISPlay:ANNotation<n>:XPOSition" on page 331
- ":DISPlay:ANNotation<n>:YPOSition" on page 332
- "Introduction to :DISPlay Commands" on page 322

:DISPlay:ANNotation<n>:GRID

 (see page 1292)

Command Syntax

```
:DISPlay:ANNotation<n>:GRID <number>

<number> ::= an integer from 1 to 4 in NR1 format.

<n> ::= an integer from 1 to 20 in NR1 format.
```

When an annotation's mode is GRID (see :DISPlay:ANNotation<n>:MODE), the :DISPlay:ANNotation<n>:GRID command makes the association to a grid.

When an annotation is associated with a grid, you can specify the annotation color using the :DISPlay:ANNotation<n>:COLor command.

Query Syntax

```
:DISPlay:ANNotation<n>:GRID?
```

The :DISPlay:ANNotation<n>:GRID? query returns the grid to which the annotation is associated.

Return Format

```
<number><NL>

<number> ::= an integer from 1 to 4 in NR1 format.
```

See Also

- ":DISPlay:ANNotation<n>:MODE" on page 328
- ":DISPlay:ANNotation<n>:COLor" on page 326
- ":DISPlay:ANNotation<n>" on page 324
- ":DISPlay:ANNotation<n>:TEXT" on page 330
- ":DISPlay:ANNotation<n>:BACKground" on page 325
- ":DISPlay:ANNotation<n>:XPOSition" on page 331
- ":DISPlay:ANNotation<n>:YPOSition" on page 332
- "Introduction to :DISPlay Commands" on page 322

:DISPlay:ANNotation<n>:MODE

N (see page 1292)

Command Syntax

```
:DISPlay:ANNotation<n>:MODE <mode>

<mode> ::= {GRID | SOURce}

<n> ::= an integer from 1 to 4 in NR1 format.
```

The :DISPlay:ANNotation<n>:MODE command specifies the annotation's association:

- GRID – The annotation is associated with a grid. Use the :DISPlay:ANNotation<n>:GRID command to select the grid.

  The annotation will remain in the grid as the grid layout changes.

  When an annotation is associated with a grid, you can specify the annotation color using the :DISPlay:ANNotation<n>:COLor command.
- SOURce – The annotation is associated with a waveform source. Use the :DISPlay:ANNotation<n>:SOURce command to select the waveform source.

  The annotation will remain with the waveform source as the waveform is moved between grids.

  When an annotation is associated with a waveform source, its color is the same as the waveform source.

Query Syntax

```
:DISPlay:ANNotation<n>:MODE?
```

The :DISPlay:ANNotation<n>:MODE? query returns the annotation mode.

Return Format

```
<mode><NL>

<mode> ::= {GRID | SOUR}
```

See Also

- ":DISPlay:ANNotation<n>:GRID" on page 327
- ":DISPlay:ANNotation<n>:COLor" on page 326
- ":DISPlay:ANNotation<n>:SOURce" on page 329
- ":DISPlay:ANNotation<n>" on page 324
- ":DISPlay:ANNotation<n>:TEXT" on page 330
- ":DISPlay:ANNotation<n>:BACKground" on page 325
- ":DISPlay:ANNotation<n>:XPOSition" on page 331
- ":DISPlay:ANNotation<n>:YPOSition" on page 332
- "Introduction to :DISPlay Commands" on page 322

:DISPlay:ANNotation<n>:SOURce

(see page 1292)

Command Syntax

```
:DISPlay:ANNotation<n>:SOURce <source>

<source> ::= {CHANnel<n> | WMEMory<n> | FUNCtion<n> | FFT | DIGital<n>}

<n> ::= an integer in NR1 format.
```

When an annotation's mode is SOURce (see :DISPlay:ANNotation<n>:MODE), the :DISPlay:ANNotation<n>:SOURce command makes the association to a waveform source.

Query Syntax

```
:DISPlay:ANNotation<n>:SOURce?
```

The :DISPlay:ANNotation<n>:SOURce? query returns the waveform source to which the annotation is associated.

Return Format

```
<source><NL>

<source> ::= {CHAN<n> | WMEM<n> | FUNC<n> | FFT | DIG<n>}
```

See Also

- ":DISPlay:ANNotation<n>:MODE" on page 328
- ":DISPlay:ANNotation<n>" on page 324
- ":DISPlay:ANNotation<n>:TEXT" on page 330
- ":DISPlay:ANNotation<n>:BACKground" on page 325
- ":DISPlay:ANNotation<n>:XPOSition" on page 331
- ":DISPlay:ANNotation<n>:YPOSition" on page 332
- "Introduction to :DISPlay Commands" on page 322

:DISPlay:ANNotation<n>:TEXT

N (see page 1292)

Command Syntax

```
:DISPlay:ANNotation<n>:TEXT <string>

<string> ::= quoted ASCII string (up to 254 characters)

<n> ::= an integer from 1 to 20 in NR1 format.
```

The :DISPlay:ANNotation<n>:TEXT command specifies the annotation string. The annotation string can contain as many characters as will fit in the Edit Annotation box on the oscilloscope's screen, up to 254 characters.

You can include a carriage return in the annotation string using the characters "\n". Note that this is not a new line character but the actual "\" (backslash) and "n" characters in the string. Carriage returns lessen the number of characters available for the annotation string.

Use :DISPlay:ANNotation<n>:TEXT "" to remotely clear the annotation text. (Two sets of quote marks without a space between them creates a NULL string.)

Query Syntax

```
:DISPlay:ANNotation<n>:TEXT?
```

The :DISPlay:ANNotation<n>:TEXT? query returns the specified annotation text.

When carriage returns are present in the annotation text, they are returned as the actual carriage return character (ASCII 0x0D).

Return Format

```
<string><NL>

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

See Also

- ":DISPlay:ANNotation<n>" on page 324
- ":DISPlay:ANNotation<n>:COLor" on page 326
- ":DISPlay:ANNotation<n>:BACKground" on page 325
- ":DISPlay:ANNotation<n>:XPOSition" on page 331
- ":DISPlay:ANNotation<n>:YPOSition" on page 332
- "Introduction to :DISPlay Commands" on page 322

:DISPlay:ANNotation<n>:XPOSition

(see page 1292)

Command Syntax

```
:DISPlay:ANNotation<n>:XPOSition <value>

<value> ::= a decimal from 0.0 to 1.0 in NR3 format that is a percent
   of the grid width.

<n> ::= an integer from 1 to 20 in NR1 format.
```

The :DISPlay:ANNotation<n>:XPOSition command sets the annotation's horizontal X position.

Query Syntax

```
:DISPlay:ANNotation<n>:XPOSition?
```

The :DISPlay:ANNotation<n>:XPOSition? query returns the annotation's horizontal X position.

Return Format

```
<value><NL>

<value> ::= a decimal from 0.0 to 1.0 in NR3 format that is a percent
   of the grid width.
```

See Also

- ":DISPlay:ANNotation<n>:YPOSition" on page 332
- ":DISPlay:ANNotation<n>" on page 324
- ":DISPlay:ANNotation<n>:COLor" on page 326
- ":DISPlay:ANNotation<n>:BACKground" on page 325
- ":DISPlay:ANNotation<n>:TEXT" on page 330

:DISPlay:ANNotation<n>:YPOSition

(see page 1292)

Command Syntax

```
:DISPlay:ANNotation<n>:YPOSition <value>

<value> ::= a decimal from 0.0 to 1.0 in NR3 format that is a percent
   of the grid height.

<n> ::= an integer from 1 to 20 in NR1 format.
```

The :DISPlay:ANNotation<n>:YPOSition command sets the annotation's vertical Y position.

Query Syntax

```
:DISPlay:ANNotation<n>:YPOSition?
```

The :DISPlay:ANNotation<n>:YPOSition? query returns the annotation's vertical Y position.

Return Format

```
<value><NL>

<value> ::= a decimal from 0.0 to 1.0 in NR3 format that is a percent
   of the grid height.
```

See Also

- ":DISPlay:ANNotation<n>:XPOSition" on page 331
- ":DISPlay:ANNotation<n>" on page 324
- ":DISPlay:ANNotation<n>:COLor" on page 326
- ":DISPlay:ANNotation<n>:BACKground" on page 325
- ":DISPlay:ANNotation<n>:TEXT" on page 330

:DISPlay:BACKlight

N (see page 1292)

Command Syntax `:DISPlay:BACKlight {{0 | OFF} | {1 | ON}}`

The :DISPlay:BACKlight command turns the display's backlight off or on.

:DISPlay:CLEar

 (see page 1292)

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" on page 322

:DISPlay:CLOCk:IGRid

 (see page 1292)

Command Syntax `:DISPlay:CLOCk:IGRid {{0 | OFF} | {1 | ON}}`

When the oscilloscope's clock information is displayed (see :DISPlay:CLOCk[:STATe]), the :DISPlay:CLOCk:IGRid command specifies whether the clock information is displayed in the first waveform grid (ON) or in the badges bar at the top of the screen (OFF).

Query Syntax `:DISPlay:CLOCk:IGRid?`

The :DISPlay:CLOCk:IGRid? query returns whether the oscilloscope's clock information is displayed in the first waveform grid (1) or in the badges bar at the top of the screen (0).

Return Format `<setting><NL>`

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

See Also

- ":DISPlay:CLOCk:IGRid:XPOSition" on page 336
- ":DISPlay:CLOCk:IGRid:YPOSition" on page 337
- ":DISPlay:CLOCk[:STATe]" on page 338

:DISPlay:CLOCk:IGRid:XPOSition

(see page 1292)

Command Syntax

```
:DISPlay:CLOCk:IGRid:XPOSition <value>

<value> ::= a decimal from 0.0 to 1.0 in NR3 format that is a percent of
 the grid width.
```

When the oscilloscope's clock information is displayed (see :DISPlay:CLOCk[:STATe]) and is located in the first waveform grid (see :DISPlay:CLOCk:IGRid), the :DISPlay:CLOCk:IGRid:XPOSition command specifies the horizontal X position of the clock information as a percent of the grid width.

Query Syntax

```
:DISPlay:CLOCk:IGRid:XPOSition?
```

The :DISPlay:CLOCk:IGRid:XPOSition? query returns the oscilloscope's clock information horizontal X position as a percent of the grid width.

Return Format

```
<value><NL>

<value> ::= a decimal from 0.0 to 1.0 in NR3 format that is a percent of
 the grid width.
```

See Also

- ":DISPlay:CLOCk:IGRid" on page 335
- ":DISPlay:CLOCk:IGRid:YPOSition" on page 337
- ":DISPlay:CLOCk[:STATe]" on page 338

:DISPlay:CLOCk:IGRid:YPOSition

N (see page 1292)

Command Syntax

```
:DISPlay:CLOCk:IGRid:YPOSition <value>

<value> ::= a decimal from 0.0 to 1.0 in NR3 format that is a percent of
 the grid height.
```

When the oscilloscope's clock information is displayed (see :DISPlay:CLOCk[:STATe]) and is located in the first waveform grid (see :DISPlay:CLOCk:IGRid), the :DISPlay:CLOCk:IGRid:YPOSition command specifies the vertical Y position of the clock information as a percent of the grid height.

Query Syntax

```
:DISPlay:CLOCk:IGRid:YPOSition?
```

The :DISPlay:CLOCk:IGRid:YPOSition? query returns the oscilloscope's clock information vertical Y position as a percent of the grid height.

Return Format

```
<value><NL>

<value> ::= a decimal from 0.0 to 1.0 in NR3 format that is a percent of
 the grid height.
```

See Also

- ":DISPlay:CLOCk:IGRid" on page 335
- ":DISPlay:CLOCk:IGRid:XPOSition" on page 336
- ":DISPlay:CLOCk[:STATe]" on page 338

:DISPlay:CLOCk[:STATe]

(see page 1292)

Command Syntax `:DISPlay:CLOCk[:STATe] {{0 | OFF} | {1 | ON}}`

The :DISPlay:CLOCk[:STATe] command enables or disables the display of the oscilloscope's clock information.

Query Syntax `:DISPlay:CLOCk[:STATe]?`

The :DISPlay:CLOCk[:STATe]? query returns whether the oscilloscope's clock information is displayed on screen.

Return Format `<setting><NL>`

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

See Also

- ":DISPlay:CLOCk:IGRid" on page 335
- ":DISPlay:CLOCk:IGRid:XPOSition" on page 336
- ":DISPlay:CLOCk:IGRid:YPOSition" on page 337

:DISPlay:DATA?

(see page 1292)

Query Syntax

```
:DISPlay:DATA? [<format>]

<format> ::= {BMP | PNG}
```

The :DISPlay:DATA? query reads screen image data. You can choose 24-bit BMP or 24-bit PNG formats.

If no format option is specified, the screen image is returned as specified by the front panel's **File > Save...** dialog box's **Format** selection.

Screen image data is returned in the IEEE-488.2 # binary block data format.

Return Format

```
<display data><NL>

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

See Also

- "Introduction to :DISPlay Commands" on page 322
- ":HCOPy:SDUMp:DATA?" on page 470
- ":HCOPy:SDUMp[:DATA]:FORMat" on page 471
- "*RCL (Recall)" on page 193
- "*SAV (Save)" on page 197
- ":VIEW" on page 227

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? PNG"
byteData = myScope.ReadIEEEBlock(BinaryType_UI1)
' Output display data to a file:
strPath = "c:\scope\data\screen.png"
' 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 f
or output.
Put #1, , byteData    ' Write data.
Close #1    ' Close file.
myScope.IO.Timeout = 5000
```

See complete example programs at: Chapter 44, “Programming Examples,” starting on page 1301

:DISPlay:GRATicule:ALABels

(see page 1292)

Command Syntax

```
:DISPlay:GRATicule:ALABels {{0 | OFF} | {1 | ON}}
```

The :DISPlay:GRATicule:ALABels command turns graticule (grid) axis labels on or off.

Query Syntax

```
:DISPlay:GRATicule:ALABels?
```

The :DISPlay:GRATicule:ALABels? query returns the graticule (grid) axis labels setting.

Return Format

```
<setting><NL>

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

See Also

- ":DISPlay:GRATicule:INTensity" on page 345
- ":DISPlay:GRATicule:ALABels:DUAL" on page 341
- ":DISPlay:GRATicule:ALABels:IGRid" on page 342

:DISPlay:GRATicule:ALABels:DUAL

N (see page 1292)

Command Syntax `:DISPlay:GRATicule:ALABels:DUAL {{0 | OFF} | {1 | ON}}`

When multiple waveforms are displayed in a grid, grid scales are displayed (see :DISPlay:GRATicule:ALABels), and scales are not shown within the grid (see :DISPlay:GRATicule:ALABels:IGRid), the :DISPlay:GRATicule:ALABels:DUAL command specifies whether grid scales are shown for two waveforms.

Query Syntax `:DISPlay:GRATicule:ALABels:DUAL?`

The :DISPlay:GRATicule:ALABels:DUAL? query returns whether grid scales are shown for two waveforms.

Return Format `<setting><NL>`

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

See Also

- ":DISPlay:GRATicule:ALABels" on page 340
- ":DISPlay:GRATicule:ALABels:IGRid" on page 342

:DISPlay:GRATicule:ALABels:IGRid

N (see page 1292)

Command Syntax `:DISPlay:GRATicule:ALABels:IGRid {{0 | OFF} | {1 | ON}}`

When grid scales are displayed (see :DISPlay:GRATicule:ALABels), the :DISPlay:GRATicule:ALABels:IGRid command specifies whether graticule (grid) axis labels are displayed in separate scale bars (OFF) or within the grid (ON).

In order to display grid axis scales for two waveforms (see :DISPlay:GRATicule:ALABels:DUAL), grid axis labels must be displayed in in separate scale bars.

Query Syntax `:DISPlay:GRATicule:ALABels:IGRid?`

The :DISPlay:GRATicule:ALABels:IGRid? query returns whether graticule (grid) axis labels are displayed in separate scale bars (0) or within the grid (1).

Return Format `<setting><NL>`

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

See Also

- ":DISPlay:GRATicule:ALABels" on page 340
- ":DISPlay:GRATicule:ALABels:DUAL" on page 341

:DISPlay:GRATicule:COUNt

N (see page 1292)

Command Syntax

```
:DISPlay:GRATicule:COUNt <num_grids>

<value> ::= an integer from 1 to 4 in NR1 format.
```

When the graticule (grid) layout is set to CUSTom (see :DISPlay:GRATicule:LAYout), the :DISPlay:GRATicule:COUNt command specifies the number of waveform grids.

Query Syntax

```
:DISPlay:GRATicule:COUNt?
```

The :DISPlay:GRATicule:COUNt? query returns the number of waveform grids.

Return Format

```
<num_grids><NL>

<num_grids> ::= an integer from 1 to 4 in NR1 format.
```

See Also

- ":DISPlay:GRATicule:LAYout" on page 346
- ":DISPlay:GRATicule:SET" on page 347
- ":DISPlay:GRATicule:SOURce?" on page 348

:DISPlay:GRATicule:FSCReen

(see page 1292)

Command Syntax

```
:DISPlay:GRATicule:FSCReen {{0 | OFF} | {1 | ON}}
```

The :DISPlay:GRATicule:FSCReen command specifies whether the waveform display grids occupy the full screen.

Query Syntax

```
:DISPlay:GRATicule:FSCReen?
```

The :DISPlay:GRATicule:FSCReen? query returns the whether the waveform display grids occupy the full screen.

Return Format

```
<setting><NL>

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

See Also

- ":DISPlay:RESults:SIZE" on page 360
- ":DISPlay:GRATicule:LAYout" on page 346

:DISPlay:GRATicule:INTensity

(see page 1292)

Command Syntax

```
:DISPlay:GRATicule:INTensity <value>

<value> ::= an integer from 0 to 100 in NR1 format.
```

The :DISPlay:GRATicule:INTensity command sets the graticule (grid) intensity.

Query Syntax

```
:DISPlay:GRATicule:INTensity?
```

The :DISPlay:GRATicule:INTensity? query returns the graticule (grid) intensity setting.

Return Format

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

See Also

- ":DISPlay:GRATicule:ALABels" on page 340

:DISPlay:GRATicule:LAYout

N (see page 1292)

Command Syntax

```
:DISPlay:GRATicule:LAYout <type>

<type> ::= {STACked | TILed | OVERlay | CUSTom}
```

The :DISPlay:GRATicule:LAYout command specifies the layout of waveform display grids:

- STACked – Waveform display grids are stacked vertically.

  Up to four grids are automatically used depending on the number of waveforms that are displayed. Assignment of waveforms to grids also happens automatically.

- TILed – Waveform display grids are tiled horizontally and vertically.

  Up to four grids are automatically used depending on the number of waveforms that are displayed. Assignment of waveforms to grids also happens automatically.

- OVERlay – All waveforms are overlaid in one grid.
- CUSTom – Specifies a custom waveform display grid layout where the number of grids can be specified (see :DISPlay:GRATicule:COUNt) and waveform sources can be assigned to particular grids (see :DISPlay:GRATicule:SET).

Query Syntax

```
:DISPlay:GRATicule:LAYout?
```

The :DISPlay:GRATicule:LAYout? query returns the waveform display grid layout.

Return Format

```
<type><NL>

<type> ::= {STAC | TIL | OVER | CUST}
```

See Also

- ":DISPlay:GRATicule:COUNt" on page 343
- ":DISPlay:GRATicule:SET" on page 347
- ":DISPlay:GRATicule:SOURce?" on page 348

:DISPlay:GRATicule:SET

(see page 1292)

Command Syntax

```
:DISPlay:GRATicule:SET <source>,<grid>

<source> ::= {CHANnel<n> | WMEMory<n> | FUNCtion<n> | FFT | SBUS<n>
   | GAIN | PHASe}

<grid> ::= a grid number inter from 1 to 4 in NR1 format.
```

When the graticule (grid) layout is set to CUSTom (see :DISPlay:GRATicule:LAYout), the :DISPlay:GRATicule:SET command assigns a waveform source to a particular grid.

The specified grid must be enabled before a waveform source can be assigned to it (see :DISPlay:GRATicule:COUNt).

To view the waveform sources assigned to a grid, use the :DISPlay:GRATicule:SOURce? query.

See Also

- ":DISPlay:GRATicule:COUNt" on page 343
- ":DISPlay:GRATicule:LAYout" on page 346
- ":DISPlay:GRATicule:SOURce?" on page 348

:DISPlay:GRATicule:SOURce?

(see page 1292)

Query Syntax

```
:DISPlay:GRATicule:SOURce? <grid>

<grid> ::= a grid number integer from 1 to 4 in NR1 format.
```

The :DISPlay:GRATicule:SOURce? query returns a comma-separated list of the waveforms in the grid.

The list can contain items like: CHAN<n>, WMEM<n>, FUNC<n>, FFT, SBUS<n>, GAIN, PHASe

When the graticule (grid) layout is set to CUSTom (see :DISPlay:GRATicule:LAYout), you can assign waveform sources to a particular grid using the :DISPlay:GRATicule:SET command.

Return Format

```
<sources><NL>

<sources> ::= comma-separated list of the waveforms displayed in the gri
d.
```

See Also

- ":DISPlay:GRATicule:COUNt" on page 343
- ":DISPlay:GRATicule:LAYout" on page 346
- ":DISPlay:GRATicule:SET" on page 347

:DISPlay:INTensity:WAVeform

(see page 1292)

Command Syntax

```
:DISPlay:INTensity:WAVeform <value>
<value> ::= an integer from 0 to 100 in NR1 format.
```

The :DISPlay:INTensity:WAVeform command sets the waveform intensity.

This is the same as adjusting the front panel **[Intensity]** knob.

Query Syntax

```
:DISPlay:INTensity:WAVeform?
```

The :DISPlay:INTensity:WAVeform? query returns the waveform intensity setting.

Return Format

```
<value><NL>
<value> ::= an integer from 0 to 100 in NR1 format.
```

See Also

- "Introduction to :DISPlay Commands" on page 322

:DISPlay:LABel

N (see page 1292)

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" on page 322
- ":CHANnel<n>:LABel" on page 278

Example Code

```
' DISP_LABEL
'  - Turns label names ON or OFF on the analyzer display.
myScope.WriteString ":DISPlay:LABel ON"    ' Turn on labels.
```

See complete example programs at: Chapter 44, “Programming Examples,” starting on page 1301

:DISPlay:LABList

N (see page 1292)

Command Syntax

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

<binary block> ::= an ordered list of up to 75 labels, a maximum of 32
                   characters each, separated by newline characters.
```

The :DISPlay:LABList command adds labels to the label list. Labels are added in alphabetical order.

NOTE

Labels that begin with the same alphabetic base string followed by decimal digits are considered duplicate labels. Duplicate labels are not added to the label list. For example, if label "A0" is in the list and you try to add a new label called "A12345678901234567890123456789012345678901", the new label is not added.

Query Syntax

```
:DISPlay:LABList?
```

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

Return Format

```
<binary block><NL>

<binary block> ::= an ordered list of up to 75 labels, a maximum of 32
                   characters each, separated by newline characters.
```

See Also

- "Introduction to :DISPlay Commands" on page 322
- ":DISPlay:LABel" on page 350
- ":CHANnel<n>:LABel" on page 278
- ":DIGital<d>:LABel" on page 315
- ":BUS<n>:LABel" on page 256

:DISPlay:MESSage:CLEar

 (see page 1292)

Command Syntax `:DISPlay:MESSage:CLEar`

The :DISPlay:MESSage:CLEar command removes all user messages that are currently on screen.

See Also

- ":SYSTem:DSP" on page 1027

:DISPlay:PERSistence

(see page 1292)

Command Syntax

```
:DISPlay:PERSistence <value>

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

<time> ::= seconds in in NR3 format from 100E-3 to 60E0
```

The :DISPlay:PERSistence command specifies the persistence setting:

- MINimum – indicates zero persistence.
- INFinite – indicates infinite persistence.
- <time> – for variable persistence, that is, you can specify how long acquisitions remain on the screen.

Use the :DISPlay:PERSistence:CLEar command to erase points stored by persistence.

NOTE Persistence is not available for math or FFT functions.

Query Syntax

```
:DISPlay:PERSistence?
```

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

Return Format

```
<value><NL>

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

See Also

- "Introduction to :DISPlay Commands" on page 322
- ":DISPlay:PERSistence:CLEar" on page 354

:DISPlay:PERSistence:CLEar

(see page 1292)

Command Syntax `:DISPlay:PERSistence:CLEar`

The :DISPlay:PERSistence:CLEar command erases all persistence data from the display, leaving the data from the last acquisition.

If the oscilloscope is running, the display will begin to accumulate waveform and persistence data again.

The :DISPlay:CLEar command clears all waveform data from the display, including the data from the last acquisition.

See Also

- ":DISPlay:PERSistence" on page 353
- ":DISPlay:CLEar" on page 334

:DISPlay:RESults:CATalog?

 (see page 1292)

Query Syntax

```
:DISPlay:RESults:CATalog?
```

The :DISPlay:RESults:CATalog? query returns a comma-separated list of the windows in the results area.

The list can contain items like: MEAS, MARK, HIST, DVM, COUN, LIST, MTES, FRAN

Return Format

```
<windows><NL>

<windows> ::= comma-separated list of the results windows displayed
   from left to right.
```

See Also

- ":DISPlay:RESults:LAYout" on page 356
- ":DISPlay:RESults:SIZE" on page 360

:DISPlay:RESults:LAYout

(see page 1292)

Command Syntax

```
:DISPlay:RESults:LAYout <format>

<format> ::= {TAB | LIST}
```

The :DISPlay:RESults:LAYout command specifies the layout of windows in the results area:

- TAB – Windows in the results area are tabbed.
- LIST – Windows in the results area are listed next to each other and buttons at the bottom of the results area can be used to highlight a particular window.

Query Syntax

```
:DISPlay:RESults:LAYout?
```

The :DISPlay:RESults:LAYout? query returns the specified results area layout.

Return Format

```
<format><NL>

<format> ::= {TAB | LIST}
```

See Also

- ":DISPlay:RESults:CATalog?" on page 355
- ":DISPlay:RESults:SIZE" on page 360
- ":DISPlay:RESults:LAYout:LIST[:SELect]" on page 357
- ":DISPlay:RESults:LAYout:TAB:LEFT[:SELect]" on page 358
- ":DISPlay:RESults:LAYout:TAB:RIGHt[:SELect]" on page 359

:DISPlay:RESults:LAYout:LIST[:SELect]

N (see page 1292)

Command Syntax

```
:DISPlay:RESults:LAYout:LIST[:SELect] <results_window>

<results_window> ::= {MEAS | MARK | HIST | DVM | COUN | LIST | MTES | FRAN}
```

When windows in the results area are listed next to each other and buttons at the bottom of the results area can be used to highlight a particular window (see :DISPlay:RESults:LAYout), the :DISPlay:RESults:LAYout:LIST[:SELect] command lets you select and highlight a particular window.

Query Syntax

```
:DISPlay:RESults:LAYout:LIST[:SELect]?
```

The :DISPlay:RESults:LAYout:LIST[:SELect]? query returns the selected and highlighted window when windows in the results area are listed next to each other.

Return Format

```
<results_window><NL>

<results_window> ::= {MEAS | MARK | HIST | DVM | COUN | LIST | MTES | FRAN}
```

See Also

- ":DISPlay:RESults:LAYout" on page 356
- ":DISPlay:RESults:LAYout:LIST[:SELect]" on page 357
- ":DISPlay:RESults:LAYout:TAB:LEFT[:SELect]" on page 358
- ":DISPlay:RESults:LAYout:TAB:RIGHt[:SELect]" on page 359

:DISPlay:RESults:LAYout:TAB:LEFT[:SELect]

 (see page 1292)

Command Syntax

```
:DISPlay:RESults:LAYout:TAB:LEFT[:SELect] <results_window>

<results_window> ::= {MEAS | MARK | HIST | LIST | MTES | FRAN}
```

When windows in the results area are tabbed (see :DISPlay:RESults:LAYout), the :DISPlay:RESults:LAYout:TAB:LEFT[:SELect] command selects a particular window to display in the left pane.

Query Syntax

```
:DISPlay:RESults:LAYout:TAB:LEFT[:SELect]?
```

The :DISPlay:RESults:LAYout:TAB:LEFT[:SELect]? query returns the left pane's selected window.

Return Format

```
<results_window><NL>

<results_window> ::= {MEAS | MARK | HIST | LIST | MTES | FRAN}
```

See Also

- ":DISPlay:RESults:LAYout" on page 356
- ":DISPlay:RESults:LAYout:LIST[:SELect]" on page 357
- ":DISPlay:RESults:LAYout:TAB:LEFT[:SELect]" on page 358
- ":DISPlay:RESults:LAYout:TAB:RIGHt[:SELect]" on page 359

:DISPlay:RESults:LAYout:TAB:RIGHt[:SELect]

N (see page 1292)

Command Syntax

```
:DISPlay:RESults:LAYout:TAB:RIGHt[:SELect] <results_window>

<results_window> ::= {DVM | COUN}
```

When windows in the results area are tabbed (see :DISPlay:RESults:LAYout), the :DISPlay:RESults:LAYout:TAB:RIGHt[:SELect] command selects a particular window to display in the right pane.

Query Syntax

```
:DISPlay:RESults:LAYout:TAB:RIGHt[:SELect]?
```

The :DISPlay:RESults:LAYout:TAB:RIGHt[:SELect]? query returns the right pane's selected window.

Return Format

```
<results_window><NL>

<results_window> ::= {DVM | COUN}
```

See Also

- ":DISPlay:RESults:LAYout" on page 356
- ":DISPlay:RESults:LAYout:LIST[:SELect]" on page 357
- ":DISPlay:RESults:LAYout:TAB:LEFT[:SELect]" on page 358
- ":DISPlay:RESults:LAYout:TAB:RIGHt[:SELect]" on page 359

:DISPlay:RESults:SIZE

(see page 1292)

Command Syntax

```
:DISPlay:RESults:SIZE <size>

<size> ::= {CUSTom,<height> | FULL | HIDDen}

<height> ::= a pixel height integer from 36 to 1024 in NR1 format.
```

The :DISPlay:RESults:SIZE command specifies the size of the results area:

- CUSTom,<height> — Lets you specify the height of the results area.
- FULL — The results area takes the full height of the display. A full height results area is 1024 pixels high.
- HIDDen — The results area is minimized so that only the titles of the windows are visible. A hidden results area is 36 pixels high.

Query Syntax

```
:DISPlay:RESults:SIZE?
```

The :DISPlay:RESults:SIZE? query returns the size of the results area.

Return Format

```
<height><NL>

<height> ::= a pixel height integer from 36 to 1024 in NR1 format.
```

See Also

- ":DISPlay:RESults:CATalog?" on page 355
- ":DISPlay:RESults:LAYout" on page 356
- ":DISPlay:GRATicule:FSCReen" on page 344

:DISPlay:TRANsparent

N (see page 1292)

Command Syntax

```
:DISPlay:TRANsparent <setting>

<setting> ::= {OFF | ON}
```

The :DISPlay:TRANsparent command enables or disables transparent mode for dialog box backgrounds in the front panel graphical user interface.

This command maps to the **Transparent Dialogs** option that appears in the graphical user interface User Options dialog box's Preferences tab (**Utilities > User Options...**).

Query Syntax

```
:DISPlay:TRANsparent?
```

The :DISPlay:TRANsparent? query returns the transparent setting.

Return Format

```
<setting><NL>

<setting> ::= {OFF | ON}
```

See Also

- "Introduction to :DISPlay Commands" on page 322

15 :DVM Commands

These commands control the digital voltmeter (DVM) feature.

Table 80 :DVM Commands Summary

Command	Query	Options and Query Returns
:DVM:ARANge {{0 | OFF} | {1 | ON}} (see page 364)	:DVM:ARANge? (see page 364)	{0 | 1}
n/a	:DVM:CURRent? (see page 365)	<dvm_value> ::= floating-point number in NR3 format
:DVM:ENABle {{0 | OFF} | {1 | ON}} (see page 366)	:DVM:ENABle? (see page 366)	{0 | 1}
:DVM:MODE <mode> (see page 367)	:DVM:MODE? (see page 367)	<dvm_mode> ::= {ACRMs | DC | DCRMs}
:DVM:SOURce <source> (see page 368)	:DVM:SOURce? (see page 368)	<source> ::= {CHANnel<n>} <n> ::= 1-2 or 1-4 in NR1 format

:DVM:ARANge

(see page 1292)

Command Syntax

```
:DVM:ARANge <setting>

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

If the selected digital voltmeter (DVM) source channel is not used in oscilloscope triggering, the :DVM:ARANge command turns the digital voltmeter's Auto Range capability on or off.

- When on, the DVM channel's vertical scale, vertical (ground level) position, and trigger (threshold voltage) level (used for the counter frequency measurement) are automatically adjusted.

  The Auto Range capability overrides attempted adjustments of the channel's vertical scale and position.
- When off, you can adjust the channel's vertical scale and position normally.

Query Syntax

```
:DVM:ARANge?
```

The :DVM:ARANge? query returns a flag indicating whether the digital voltmeter's Auto Range capability is on or off.

Return Format

```
<setting><NL>

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

See Also

- ":DVM:SOURce" on page 368
- ":DVM:ENABle" on page 366
- ":DVM:MODE" on page 367

:DVM:CURRent?

(see page 1292)

Query Syntax `:DVM:CURRent?`

The :DVM:CURRent? query returns the displayed 3-digit DVM value based on the current mode.

NOTE It can take up to a few seconds after DVM analysis is enabled before this query starts to produce good results, that is, results other than +9.9E+37. To wait for good values after DVM analysis is enabled, programs should loop until a value less than +9.9E+37 is returned.

Return Format

```
<dvm_value><NL>

<dvm_value> ::= floating-point number in NR3 format
```

See Also

- ":DVM:SOURce" on page 368
- ":DVM:ENABle" on page 366
- ":DVM:MODE" on page 367

:DVM:ENABle

(see page 1292)

Command Syntax

```
:DVM:ENABle <setting>

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

The :DVM:ENABle command turns the digital voltmeter (DVM) analysis feature on or off.

Query Syntax

```
:DVM:ENABle?
```

The :DVM:ENABle? query returns a flag indicating whether the digital voltmeter (DVM) analysis feature is on or off.

Return Format

```
<setting><NL>

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

See Also

- ":DVM:SOURce" on page 368
- ":DVM:MODE" on page 367
- ":DVM:ARANge" on page 364

:DVM:MODE

(see page 1292)

Command Syntax

```
:DVM:MODE <dvm_mode>

<dvm_mode> ::= {ACRMs | DC | DCRMs}
```

The :DVM:MODE command sets the digital voltmeter (DVM) mode:

- ACRMs — displays the root-mean-square value of the acquired data, with the DC component removed.
- DC — displays the DC value of the acquired data.
- DCRMs — displays the root-mean-square value of the acquired data.

Query Syntax

```
:DVM:MODE?
```

The :DVM:MODE? query returns the selected DVM mode.

Return Format

```
<dvm_mode><NL>

<dvm_mode> ::= {ACRM | DC | DCRM}
```

See Also

- ":DVM:ENABle" on page 366
- ":DVM:SOURce" on page 368
- ":DVM:ARANge" on page 364
- ":DVM:CURRent?" on page 365
- ":TRIGger:MODE" on page 1091

:DVM:SOURce

N (see page 1292)

Command Syntax

```
:DVM:SOURce <source>

<source> ::= {CHANnel<n>}

<n> ::= 1-2 or 1-4 in NR1 format
```

The :DVM:SOURce command sets the select the analog channel on which digital voltmeter (DVM) measurements are made.

The selected channel does not have to be on (displaying a waveform) in order for DVM measurements to be made.

Query Syntax

```
:DVM:SOURce?
```

The :DVM:SOURce? query returns the selected DVM input source.

Return Format

```
<source><NL>

<source> ::= {CHAN<n>}

<n> ::= 1-2 or 1-4 in NR1 format
```

See Also

- ":DVM:ENABle" on page 366
- ":DVM:MODE" on page 367
- ":DVM:ARANge" on page 364
- ":DVM:CURRent?" on page 365

16 :EXTernal Trigger Commands

Control the input characteristics of the external trigger input. See "Introduction to :EXTernal Trigger Commands" on page 369.

Table 81 :EXTernal Trigger Commands Summary

Command	Query	Options and Query Returns
:EXTernal:BWLimit <bwlimit> (see page 370)	:EXTernal:BWLimit? (see page 370)	<bwlimit> ::= {0 | OFF}
:EXTernal:PROBe <attenuation> (see page 371)	:EXTernal:PROBe? (see page 371)	<attenuation> ::= probe attenuation ratio in NR3 format
:EXTernal:RANGe <range>[<suffix>] (see page 372)	:EXTernal:RANGe? (see page 372)	<range> ::= vertical full-scale range value in NR3 format <suffix> ::= {V | mV}
:EXTernal:UNITs <units> (see page 373)	:EXTernal:UNITs? (see page 373)	<units> ::= {VOLT | AMPere}

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;RANG +8.0E+00;UNIT VOLT;PROB +1.0000000E+00
```

:EXTernal:BWLimit

 (see page 1292)

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" on page 369
- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:HFReject" on page 1082

:EXTernal:PROBe

 (see page 1292)

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 can be set from 0.001 to 10000 in a 1-2-5 sequence. 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" on page 369
- ":EXTernal:RANGe" on page 372
- "Introduction to :TRIGger Commands" on page 1077
- ":CHANnel<n>:PROBe" on page 280

:EXTernal:RANGe

(see page 1292)

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. When using 1:1 probe attenuation, the range can only be set to 8.0 V.

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
```

See Also

- "Introduction to :EXTernal Trigger Commands" on page 369
- ":EXTernal:PROBe" on page 371
- "Introduction to :TRIGger Commands" on page 1077

:EXTernal:UNITs

(see page 1292)

Command Syntax

```
:EXTernal:UNITs <units>

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

The :EXTernal:UNITs command sets the measurement units for the probe connected to the external trigger input. Select VOLT for a voltage probe and select AMPere 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" on page 369
- "Introduction to :TRIGger Commands" on page 1077
- ":EXTernal:RANGe" on page 372
- ":EXTernal:PROBe" on page 371
- ":CHANnel<n>:UNITs" on page 297

17 :FFT Commands

Control the FFT function in the oscilloscope. See "Introduction to :FFT Commands" on page 377.

Table 82 :FFT Commands Summary

Command	Query	Options and Query Returns
:FFT:AVERage:COUNt <count> (see page 378)	:FFT:AVERage:COUNt? (see page 378)	<count> ::= an integer from 2 to 65536 in NR1 format.
n/a	:FFT:BSIZe? (see page 379)	<bin_size> ::= the FFT resolution as a bin size frequency width in NR3 format.
:FFT:CENTer <frequency> (see page 380)	:FFT:CENTer? (see page 380)	<frequency> ::= the current center frequency in NR3 format. The range of legal values is from -25 GHz to 25 GHz.
:FFT:CLEar (see page 381)	n/a	n/a
:FFT:DETection:POINts <pts_per_span> (see page 382)	:FFT:DETection:POINts? (see page 382)	<pts_per_span> ::= an integer from 640 to 65536 in NR1 format.
:FFT:DETection:TYPE <detection> (see page 383)	:FFT:DETection:TYPE? (see page 383)	<detection> ::= {OFF | SAMPle | PPOSitive | PNEGative | AVERage | NORMal}
:FFT:DISPlay {{0 | OFF} | {1 | ON}} (see page 384)	:FFT:DISPlay? (see page 384)	<s> ::= 1-6, in NR1 format. {0 | 1}
:FFT:DMODe <display_mode> (see page 385)	:FFT:DMODe? (see page 385)	<display_mode> ::= {NORMal | AVERage | MAXHold | MINHold}
:FFT:FREQuency:STARt <frequency> (see page 386)	:FFT:FREQuency:STARt? (see page 386)	<frequency> ::= the start frequency in NR3 format.

Table 82 :FFT Commands Summary (continued)

Command	Query	Options and Query Returns
:FFT:FREQuency:STOP <frequency> (see page 387)	:FFT:FREQuency:STOP? (see page 387)	<frequency> ::= the stop frequency in NR3 format.
:FFT:GATE <gating> (see page 388)	:FFT:GATE? (see page 388)	<gating> ::= {NONE | ZOOM}
:FFT:OFFSet <offset> (see page 389)	:FFT:OFFSet? (see page 389)	<offset> ::= the value at center screen in NR3 format.
:FFT:RANGe <range> (see page 390)	:FFT:RANGe? (see page 390)	<range> ::= the full-scale vertical axis value in NR3 format.
n/a	:FFT:RBWidth? (see page 391)	<resolution_bw> ::= the FFT resolution as a resolution bandwidth frequency in NR3 format.
:FFT:READout <type> (see page 392)	:FFT:READout? (see page 392)	<type> ::= {OFF | SRATe | BSIZe | RBWidth}
:FFT:REFerence <level> (see page 393)	:FFT:REFerence? (see page 393)	<level> ::= the current reference level in NR3 format.
:FFT:SCALe <scale value>[<suffix>] (see page 394)	:FFT:SCALe? (see page 394)	<scale_value> ::= integer in NR1 format. <suffix> ::= dB
:FFT:SOURce <source> (see page 395)	:FFT:SOURce? (see page 395)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m>} <n> ::= 1 to (# analog channels) in NR1 format. <m> ::= 1 to (# math functions) in NR1 format
:FFT:SPAN (see page 396)	:FFT:SPAN? (see page 396)	::= the current frequency span in NR3 format. Legal values are 1 Hz to 100 GHz.
n/a	:FFT:SRATe? (see page 397)	<sample_rate> ::= the FFT resolution as a sample rate frequency in NR3 format.
:FFT:VTYPe <units> (see page 398)	:FFT:VTYPe? (see page 398)	<units> ::= {DECibel | VRMS}
:FFT:WINDow <window> (see page 399)	:FFT:WINDow? (see page 399)	<window> ::= {RECTangular | HANNing | FLATtop | BHARris | BARTlett}

Introduction to :FFT Commands

The FFT subsystem controls the FFT function in the oscilloscope.

Reporting the Setup

Use :FFT? to query setup information for the FFT subsystem.

Return Format

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

```
:FFT:DISP 0;SOUR1 CHAN1;RANG +160E+00;OFFS -60.0000E+00;
SPAN +100.0000E+03;CENT +50.000000E+03;WIND HANN;VTYP DEC;AVER:COUN 8
```

:FFT:AVERage:COUNt

N (see page 1292)

Command Syntax

```
:FFT:AVERage:COUNt <count>

<count> ::= an integer from 2 to 65536 in NR1 format.
```

The :FFT:AVERage:COUNt command sets the number of waveforms to be averaged together.

The number of averages can be set from 2 to 65536 in increments of powers of 2.

Increasing the number of averages will increase resolution and reduce noise.

Query Syntax

```
:FFT:AVERage:COUNt?
```

The :FFT:AVERage:COUNt? query returns the number of waveforms to be averaged together.

Return Format

```
<count><NL>

<count> ::= an integer from 2 to 65536 in NR1 format.
```

See Also

- ":FFT:CENTer" on page 380
- ":FFT:CLEar" on page 381
- ":FFT:DISPlay" on page 384
- ":FFT:OFFSet" on page 389
- ":FFT:RANGe" on page 390
- ":FFT:REFerence" on page 393
- ":FFT:SCALe" on page 394
- ":FFT:SOURce" on page 395
- ":FFT:SPAN" on page 396
- ":FFT:FREQuency:STARt" on page 386
- ":FFT:FREQuency:STOP" on page 387
- ":FFT:VTYPe" on page 398
- ":FFT:WINDow" on page 399

:FFT:BSIZe?

(see page 1292)

Query Syntax `:FFT:BSIZe?`

The :FFT:BSIZe? query returns the FFT's bin size value.

Return Format `<bin_size><NL>`

`<bin_size> ::= the FFT resolution as a bin size frequency width in NR3 format.`

See Also

- ":FFT:RBWidth?" on page 391
- ":FFT:READout" on page 392
- ":FFT:SRATe?" on page 397

:FFT:CENTer

N (see page 1292)

Command Syntax

```
:FFT:CENTer <frequency>

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

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

Query Syntax

```
:FFT:CENTer?
```

The :FFT: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 -25 GHz to 25 GHz.
```

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

See Also

- ":FFT:AVERage:COUNt" on page 378
- ":FFT:CENTer" on page 380
- ":FFT:DISPlay" on page 384
- ":FFT:OFFSet" on page 389
- ":FFT:RANGe" on page 390
- ":FFT:REFerence" on page 393
- ":FFT:SCALe" on page 394
- ":FFT:SOURce" on page 395
- ":FFT:SPAN" on page 396
- ":FFT:FREQuency:STARt" on page 386
- ":FFT:FREQuency:STOP" on page 387
- ":FFT:VTYPe" on page 398
- ":FFT:WINDow" on page 399

:FFT:CLEar

(see page 1292)

Command Syntax `:FFT:CLEar`

When the FFT display mode is AVERage, MAXHold, or MINHold, the :FFT:CLEar command clears the number of evaluated waveforms.

See Also

- ":FFT:AVERage:COUNt" on page 378
- ":FFT:CENTer" on page 380
- ":FFT:DISPlay" on page 384
- ":FFT:OFFSet" on page 389
- ":FFT:RANGe" on page 390
- ":FFT:REFerence" on page 393
- ":FFT:SCALe" on page 394
- ":FFT:SOURce" on page 395
- ":FFT:SPAN" on page 396
- ":FFT:FREQuency:STARt" on page 386
- ":FFT:FREQuency:STOP" on page 387
- ":FFT:VTYPe" on page 398
- ":FFT:WINDow" on page 399

:FFT:DETection:POINts

N (see page 1292)

Command Syntax

```
:FFT:DETection:POINts <pts_per_span>

<pts_per_span> ::= an integer from 640 to 65536 in NR1 format.
```

For FFT (Magnitude) functions, when FFT detectors are used (see :FFT:DETection:TYPE), the :FFT:DETection:POINts command specifies the maximum number of points that detectors should decimate to. This is also the number of buckets that sampled FFT points are grouped into before the selected detection type reduction (decimation) is applied.

Query Syntax

```
:FFT:DETection:POINts?
```

The :FFT:DETection:POINts? query returns the specified maximum number of points that detectors should decimate to.

Return Format

```
<pts_per_span><NL>

<pts_per_span> ::= an integer from 640 to 65536 in NR1 format.
```

See Also

- ":FFT:DETection:TYPE" on page 383

:FFT:DETection:TYPE

N (see page 1292)

Command Syntax

```
:FFT:DETection:TYPE <detection>

<detection> ::= {OFF | SAMPle | PPOSitive | PNEGative | AVERage | NORMal
}
```

For FFT (Magnitude) functions, the :FFT:DETection:TYPE command lets you set the FFT detector decimation type.

Detectors give you a way of manipulating the acquired data to emphasize different features of the data. Detectors reduce (decimate) the number of FFT points to at most the number of specified detector points. In this reduction, sampled FFT points are bucketized, that is, split into a number of groups that equals the specified number of detector points. Then, the points in each bucket are reduced to a single point according to the selected detection type. The detector types are:

- OFF – No detector is used.
- SAMPle – Takes the point nearest to the center of every bucket.
- PPOSitive (+ Peak) – Takes the most positive point in every bucket.
- PNEGative (- Peak) – Takes the most negative point in every bucket.
- AVERage – Takes the average of all points in every bucket.
- NORMal – Implements a rosenfell algorithm. This method picks either the minimum or maximum sample in every bucket depending on whether the data is monotonically increasing, decreasing, or varying.

When detectors are used, the FFT's output is decimated, and any analysis is performed on the reduced or detected data set.

Query Syntax

```
:FFT:DETection:TYPE?
```

The :FFT:DETection:TYPE? query returns ...

Return Format

```
<detection><NL>

<detection> ::= {OFF | SAMP | PPOS | PNEG | AVER | NORM}
```

See Also

- ":FFT:DETection:POINts" on page 382

:FFT:DISPlay

N (see page 1292)

Command Syntax

```
:FFT:DISPlay {{0 | OFF} | {1 | ON}}
```

The :FFT:DISPlay command turns the display of the FFT function on or off.

When ON is selected, the FFT function is calculated and displayed.

When OFF is selected, the FFT function is neither calculated nor displayed.

Query Syntax

```
:FFT:DISPlay?
```

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

Return Format

```
<display><NL>

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

See Also

- ":FFT:AVERage:COUNt" on page 378
- ":FFT:CENTer" on page 380
- ":FFT:CLEar" on page 381
- ":FFT:OFFSet" on page 389
- ":FFT:RANGe" on page 390
- ":FFT:REFerence" on page 393
- ":FFT:SCALe" on page 394
- ":FFT:SOURce" on page 395
- ":FFT:SPAN" on page 396
- ":FFT:FREQuency:STARt" on page 386
- ":FFT:FREQuency:STOP" on page 387
- ":FFT:VTYPe" on page 398
- ":FFT:WINDow" on page 399

:FFT:DMODe

(see page 1292)

Command Syntax

```
:FFT:DMODe <display_mode>

<display_mode> ::= {NORMal | AVERage | MAXHold | MINHold}
```

The :FFT:DMODe command selects one of these FFT waveform display modes:

- NORMal – this is the FFT waveform without any averaging or hold functions applied. This is how FFT math function waveforms are displayed.
- AVERage – the FFT waveform is averaged the selected number of times. Averages are calculated using a "decaying average" approximation, where:

  next_average = current_average + (new_data - current_average)/N

  Where N starts at 1 for the first acquisition and increments for each following acquisition until it reaches the selected number of averages, where it holds.
- MAXHold – records the maximum vertical values found at each horizontal bucket across multiple analysis cycles and uses those values to build a waveform. This display mode is often referred to as Max Envelope.
- MINHold – records the minimum vertical values found at each horizontal bucket across multiple analysis cycles and uses those values to build a waveform. This display mode is often referred to as Min Envelope.

Query Syntax

```
:FFT:DMODe?
```

The :FFT:DMODe? query returns the selected FFT display mode.

Return Format

```
<display_mode><NL>

<display_mode> ::= {NORMal | AVERage | MAXHold | MINHold}
```

See Also

- ":FFT:READout" on page 392

:FFT:FREQuency:STARt

(see page 1292)

Command Syntax

```
:FFT:FREQuency:STARt <frequency>

<frequency> ::= the start frequency in NR3 format.
```

The :FFT:FREQuency:STARt command sets the start frequency in the FFT (Fast Fourier Transform) math function's displayed range.

The FFT (Fast Fourier Transform) math function's displayed range can also be set with the :FFT:CENTer and :FFT:SPAN commands.

Query Syntax

```
:FFT:FREQuency:STARt?
```

The :FFT:FREQuency:STARt? query returns the current start frequency in Hertz.

Return Format

```
<frequency><NL>

<frequency> ::= the start frequency in NR3 format.
```

See Also

- ":FFT:AVERage:COUNt" on page 378
- ":FFT:CENTer" on page 380
- ":FFT:CLEar" on page 381
- ":FFT:DISPlay" on page 384
- ":FFT:OFFSet" on page 389
- ":FFT:RANGe" on page 390
- ":FFT:REFerence" on page 393
- ":FFT:SCALe" on page 394
- ":FFT:SOURce" on page 395
- ":FFT:SPAN" on page 396
- ":FFT:FREQuency:STOP" on page 387
- ":FFT:VTYPe" on page 398
- ":FFT:WINDow" on page 399

:FFT:FREQuency:STOP

(see page 1292)

Command Syntax

```
:FFT:FREQuency:STOP <frequency>

<frequency> ::= the stop frequency in NR3 format.
```

The :FFT:FREQuency:STOP command sets the stop frequency in the FFT (Fast Fourier Transform) math function's displayed range.

The FFT (Fast Fourier Transform) math function's displayed range can also be set with the :FFT:CENTer and :FFT:SPAN commands.

Query Syntax

```
:FFT:FREQuency:STOP?
```

The :FFT:FREQuency:STOP? query returns returns the current stop frequency in Hertz.

Return Format

```
<frequency><NL>

<frequency> ::= the stop frequency in NR3 format.
```

See Also

- ":FFT:AVERage:COUNt" on page 378
- ":FFT:CENTer" on page 380
- ":FFT:CLEar" on page 381
- ":FFT:DISPlay" on page 384
- ":FFT:OFFSet" on page 389
- ":FFT:RANGe" on page 390
- ":FFT:REFerence" on page 393
- ":FFT:SCALe" on page 394
- ":FFT:SOURce" on page 395
- ":FFT:SPAN" on page 396
- ":FFT:FREQuency:STARt" on page 386
- ":FFT:VTYPe" on page 398
- ":FFT:WINDow" on page 399

:FFT:GATE

N (see page 1292)

Command Syntax

```
:FFT:GATE <gating>

<gating> ::= {NONE | ZOOM}
```

The :FFT:GATE command specifies whether the FFT is performed on the Main time base window (NONE) or the ZOOM window when the zoomed time base is displayed.

Query Syntax

```
:FFT:GATE?
```

The :FFT:GATE? query returns the gate setting.

Return Format

```
<gating><NL>

<gating> ::= {NONE | ZOOM}
```

See Also

- ":FFT:VTYPe" on page 398
- ":FFT:WINDow" on page 399
- "Introduction to :FFT Commands" on page 377

:FFT:OFFSet

(see page 1292)

Command Syntax

```
:FFT:OFFSet <offset>

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

The :FFT:OFFSet command specifies the FFT vertical value represented at center screen.

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 :FFT:OFFSet command is equivalent to the :FFT:REFerence command.**

Query Syntax

```
:FFT:OFFSet?
```

The :FFT:OFFSet? query returns the current offset value for the FFT function.

Return Format

```
<offset><NL>

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

See Also

- ":FFT:AVERage:COUNt" on page 378
- ":FFT:CENTer" on page 380
- ":FFT:CLEar" on page 381
- ":FFT:DISPlay" on page 384
- ":FFT:RANGe" on page 390
- ":FFT:REFerence" on page 393
- ":FFT:SCALe" on page 394
- ":FFT:SOURce" on page 395
- ":FFT:SPAN" on page 396
- ":FFT:FREQuency:STARt" on page 386
- ":FFT:FREQuency:STOP" on page 387
- ":FFT:VTYPe" on page 398
- ":FFT:WINDow" on page 399

:FFT:RANGe

N (see page 1292)

Command Syntax

```
:FFT:RANGe <range>

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

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

Query Syntax

```
:FFT:RANGe?
```

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

Return Format

```
<range><NL>

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

See Also

- ":FFT:AVERage:COUNt" on page 378
- ":FFT:CENTer" on page 380
- ":FFT:CLEar" on page 381
- ":FFT:DISPlay" on page 384
- ":FFT:OFFSet" on page 389
- ":FFT:REFerence" on page 393
- ":FFT:SCALe" on page 394
- ":FFT:SOURce" on page 395
- ":FFT:SPAN" on page 396
- ":FFT:FREQuency:STARt" on page 386
- ":FFT:FREQuency:STOP" on page 387
- ":FFT:VTYPe" on page 398
- ":FFT:WINDow" on page 399

:FFT:RBWidth?

(see page 1292)

Query Syntax `:FFT:RBWidth?`

The :FFT:RBWidth? query returns the FFT's resolution bandwidth value.

Return Format `<resolution_bw><NL>`

`<resolution_bw> ::= the FFT resolution as a resolution bandwidth frequency in NR3 format.`

See Also

- ":FFT:BSIZe?" on page 379
- ":FFT:READout" on page 392
- ":FFT:SRATe?" on page 397

:FFT:READout

N (see page 1292)

Command Syntax

```
:FFT:READout <type>

<type> ::= {OFF | SRATe | BSIZe | RBWidth}
```

The :FFT:READout command specifies how the FFT resolution is displayed in the FFT spectrum plot:

- OFF – the FFT resolution is not displayed.
- SRATe – the sample rate is displayed. To return the FFT's sample rate value use :FFT:SRATe?.
- BSIZe – the bin size is displayed. To return the FFT's bin size value use :FFT:BSIZe?.
- RBWidth – the resolution bandwidth is displayed. To return the FFT's resolution bandwidth value use :FFT:RBWidth?.

Query Syntax

```
:FFT:READout?
```

The :FFT:READout? query returns the selected FFT resolution display option.

Return Format

```
<type><NL>

<type> ::= {OFF | SRAT | BSIZ | RBW}
```

See Also

- ":FFT:BSIZe?" on page 379
- ":FFT:RBWidth?" on page 391
- ":FFT:SRATe?" on page 397

:FFT:REFerence

 (see page 1292)

Command Syntax

```
:FFT:REFerence <level>

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

The :FFT:REFerence command specifies the FFT vertical value represented at center screen.

If you set the reference level to a value outside of the legal range, the level is automatically set to the nearest legal value.

NOTE **The :FFT:REFerence command is equivalent to the :FFT:OFFSet command.**

Query Syntax

```
:FFT:REFerence?
```

The :FFT:REFerence? query returns the current reference level value for the FFT function.

Return Format

```
<level><NL>

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

See Also

- ":FFT:AVERage:COUNt" on page 378
- ":FFT:CENTer" on page 380
- ":FFT:CLEar" on page 381
- ":FFT:DISPlay" on page 384
- ":FFT:OFFSet" on page 389
- ":FFT:RANGe" on page 390
- ":FFT:SCALe" on page 394
- ":FFT:SOURce" on page 395
- ":FFT:SPAN" on page 396
- ":FFT:FREQuency:STARt" on page 386
- ":FFT:FREQuency:STOP" on page 387
- ":FFT:VTYPe" on page 398
- ":FFT:WINDow" on page 399

:FFT:SCALe

N (see page 1292)

Command Syntax

```
:FFT:SCALe <scale_value>[<suffix>]

<scale_value> ::= floating-point value in NR3 format.

<suffix> ::= dB
```

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

Query Syntax

```
:FFT:SCALe?
```

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

Return Format

```
<scale_value><NL>

<scale_value> ::= floating-point value in NR3 format.
```

See Also

- ":FFT:AVERage:COUNt" on page 378
- ":FFT:CENTer" on page 380
- ":FFT:CLEar" on page 381
- ":FFT:DISPlay" on page 384
- ":FFT:OFFSet" on page 389
- ":FFT:RANGe" on page 390
- ":FFT:REFerence" on page 393
- ":FFT:SOURce" on page 395
- ":FFT:SPAN" on page 396
- ":FFT:FREQuency:STARt" on page 386
- ":FFT:FREQuency:STOP" on page 387
- ":FFT:VTYPe" on page 398
- ":FFT:WINDow" on page 399

:FFT:SOURce

(see page 1292)

Command Syntax

```
:FFT:SOURce <offset>
<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m>}
<n> ::= 1 to (# analog channels) in NR1 format.
<m> ::= 1 to (# math functions) in NR1 format
```

The :FFT:SOURce command selects the source for the FFT function.

NOTE To allow similarity to the :FUNCtion<m>:SOURce1 command/query, you can also use :FFT:SOURce1 for this command and query.

Query Syntax

```
:FFT:SOURce?
```

The :FFT:SOURce? query returns the current source1 for the FFT function.

Return Format

```
<source><NL>
<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m>}
<n> ::= 1 to (# analog channels) in NR1 format.
<m> ::= 1 to (# math functions) in NR1 format
```

See Also

- ":FFT:AVERage:COUNt" on page 378
- ":FFT:CENTer" on page 380
- ":FFT:CLEar" on page 381
- ":FFT:DISPlay" on page 384
- ":FFT:OFFSet" on page 389
- ":FFT:RANGe" on page 390
- ":FFT:REFerence" on page 393
- ":FFT:SCALe" on page 394
- ":FFT:SPAN" on page 396
- ":FFT:FREQuency:STARt" on page 386
- ":FFT:FREQuency:STOP" on page 387
- ":FFT:VTYPe" on page 398
- ":FFT:WINDow" on page 399

:FFT:SPAN

N (see page 1292)

Command Syntax

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

Query Syntax

```
:FFT:SPAN?
```

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

NOTE After a *RST (Reset) or :AUToscale command, the values returned by the :FFT:CENTer? and :FFT:SPAN? queries depend on the current :TIMebase:RANGe value. Once you change either the :FFT:CENTer or :FFT: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

- ":FFT:AVERage:COUNt" on page 378
- ":FFT:CENTer" on page 380
- ":FFT:CLEar" on page 381
- ":FFT:DISPlay" on page 384
- ":FFT:OFFSet" on page 389
- ":FFT:RANGe" on page 390
- ":FFT:REFerence" on page 393
- ":FFT:SCALe" on page 394
- ":FFT:SOURce" on page 395
- ":FFT:FREQuency:STARt" on page 386
- ":FFT:FREQuency:STOP" on page 387
- ":FFT:VTYPe" on page 398
- ":FFT:WINDow" on page 399

:FFT:SRATe?

 (see page 1292)

Query Syntax `:FFT:SRATe?`

The :FFT:SRATe? query returns the FFT's sample rate value.

Return Format `<sample_rate><NL>`

`<sample_rate> ::= the FFT resolution as a sample rate frequency in NR3 format.`

See Also

- ":FFT:BSIZe?" on page 379
- ":FFT:RBWidth?" on page 391
- ":FFT:READout" on page 392

:FFT:VTYPe

(see page 1292)

Command Syntax

```
:FFT:VTYPe <units>

<units> ::= {DECibel | VRMS}
```

The :FFT:VTYPe command specifies FFT vertical units as DECibel or VRMS.

Query Syntax

```
:FFT:VTYPe?
```

The :FFT:VTYPe? query returns the current FFT vertical units.

Return Format

```
<units><NL>

<units> ::= {DEC | VRMS}
```

See Also

- ":FFT:AVERage:COUNt" on page 378
- ":FFT:CENTer" on page 380
- ":FFT:CLEar" on page 381
- ":FFT:DISPlay" on page 384
- ":FFT:OFFSet" on page 389
- ":FFT:RANGe" on page 390
- ":FFT:REFerence" on page 393
- ":FFT:SCALe" on page 394
- ":FFT:SOURce" on page 395
- ":FFT:SPAN" on page 396
- ":FFT:FREQuency:STARt" on page 386
- ":FFT:FREQuency:STOP" on page 387
- ":FFT:WINDow" on page 399

:FFT:WINDow

(see page 1292)

Command Syntax

```
:FFT:WINDow <window>

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

The :FFT:WINDow command allows the selection of 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.

- RECTangular – useful for transient signals, and signals where there are an integral number of cycles in the time record.
- HANNing – 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.
- FLATtop – best for making accurate amplitude measurements of frequency peaks.
- BHARris (Blackman-Harris) – reduces time resolution compared to the rectangular window, but it improves the capacity to detect smaller impulses due to lower secondary lobes (provides minimal spectral leakage).
- BARTlett – (triangular, with end points at zero) window is similar to the Hanning window in that it is good for making accurate frequency measurements, but its higher and wider secondary lobes make it not quite as good for resolving frequencies that are close together.

Query Syntax

```
:FFT:WINDow?
```

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

Return Format

```
<window><NL>

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

See Also

- ":FFT:AVERage:COUNt" on page 378
- ":FFT:CENTer" on page 380
- ":FFT:CLEar" on page 381
- ":FFT:DISPlay" on page 384
- ":FFT:OFFSet" on page 389

18 :FRANalysis Commands

Control oscilloscope functions associated with the Frequency Response Analysis (FRA) feature, which is available in oscilloscope models that have a license-enabled built-in waveform generator. See "Introduction to :FRANalysis Commands" on page 402.

Table 83 :FRANalysis Commands Summary

Command	Query	Options and Query Returns
n/a	:FRANalysis:DATA? [SWEep | SINGle] (see page 403)	<binary_block> ::= comma-separated data with newlines at the end of each row
:FRANalysis:ENABle {{0 | OFF} | {1 | ON}} (see page 404)	:FRANalysis:ENABle? (see page 404)	{0 | 1}
:FRANalysis:FREQuency :MODE <setting> (see page 405)	:FRANalysis:FREQuency :MODE? (see page 405)	<setting> ::= {SWEep | SINGle}
:FRANalysis:FREQuency :SINGle <value>[suffix] (see page 406)	:FRANalysis:FREQuency :SINGle? (see page 406)	<value> ::= {20 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000 | 2000000} [suffix] ::= {Hz | kHz| MHz}
:FRANalysis:FREQuency :STARt <value>[suffix] (see page 407)	:FRANalysis:FREQuency :STARt? (see page 407)	<value> ::= {20 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000} [suffix] ::= {Hz | kHz| MHz}
:FRANalysis:FREQuency :STOP <value>[suffix] (see page 408)	:FRANalysis:FREQuency :STOP? (see page 408)	<value> ::= {100 | 1000 | 10000 | 100000 | 1000000 | 10000000 | 20000000} [suffix] ::= {Hz | kHz| MHz}
:FRANalysis:PPDecade <value> (see page 409)	:FRANalysis:PPDecade? (see page 409)	<value> ::= {10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100}
:FRANalysis:RUN (see page 410)	n/a	n/a

Table 83 :FRANalysis Commands Summary (continued)

Command	Query	Options and Query Returns
:FRANalysis:SOURce:INPut <source> (see page 411)	:FRANalysis:SOURce:INPut? (see page 411)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:FRANalysis:SOURce:OUTPut <source> (see page 412)	:FRANalysis:SOURce:OUTPut? (see page 412)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:FRANalysis:TRACe <selection> (see page 413)	:FRANalysis:TRACe? (see page 413)	<selection> ::= {NONE | ALL | GAIN | PHASe}[,{GAIN | PHASe}]
:FRANalysis:WGEN:LOAD <impedance> (see page 414)	:FRANalysis:WGEN:LOAD? (see page 414)	<impedance> ::= {ONEMeg | FIFTy}
:FRANalysis:WGEN:VOLTage <amplitude>,[<range>] (see page 415)	:FRANalysis:WGEN:VOLTage? [<range>] (see page 415)	<amplitude> ::= amplitude in volts in NR3 format <range> ::= {F20HZ | F100HZ | F1KHZ | F10KHZ | F100KHZ | F1MHZ | F10MHZ | F20MHZ}
:FRANalysis:WGEN:VOLTage:PROFile {{0 | OFF} | {1 | ON}} (see page 416)	:FRANalysis:WGEN:VOLTage:PROFile? (see page 416)	{0 | 1}

Introduction to :FRANalysis Commands

The FRANalysis subsystem controls the Frequency Response Analysis feature in the oscilloscope.

The Frequency Response Analysis (FRA) feature controls the built-in waveform generator to sweep a sine wave across a range of frequencies while measuring the input to and output from a device under test (DUT). At each frequency, gain (A) and phase are measured and plotted on a frequency response chart.

Reporting the Setup

Use :FRANalysis? to query setup information for the FRANalysis subsystem.

Return Format

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

```
:FRAN:SOUR:INP CHAN1;OUTP CHAN2;:FRAN:FREQ:STAR +100E+00;
STOP +20.000000E+06;:FRAN:WGEN:VOLT +200.0E-03;LOAD FIFT
```

:FRANalysis:DATA?

(see page 1292)

Query Syntax

```
:FRANalysis:DATA? [SWEep | SINGle]
```

The :FRANalysis:DATA? query returns the frequency response analysis data.

The data is returned in four comma-separated columns of data for each step in the sweep: Frequency (Hz), Amplitude (Vpp), Gain (dB), and Phase (°).

You can use the :FRANalysis:TRACe command to specify whether to include gain, phase, both gain and phase, or neither in the frequency response analysis results.

The SWEep or SINGle option specifies whether to get the data from a sweep or single-frequency analysis (see :FRANalysis:FREQuency:MODE). If this option is not specified, the data from the sweep analysis is returned by default.

Return Format

```
<binary_block><NL>

<binary_block> ::= comma-separated data with newlines at the end of each
 row
```

See Also

- ":FRANalysis:ENABle" on page 404
- ":FRANalysis:FREQuency:MODE" on page 405
- ":FRANalysis:FREQuency:SINGle" on page 406
- ":FRANalysis:FREQuency:STARt" on page 407
- ":FRANalysis:FREQuency:STOP" on page 408
- ":FRANalysis:RUN" on page 410
- ":FRANalysis:SOURce:INPut" on page 411
- ":FRANalysis:SOURce:OUTPut" on page 412
- ":FRANalysis:TRACe" on page 413
- ":FRANalysis:WGEN:LOAD" on page 414
- ":FRANalysis:WGEN:VOLTage" on page 415

:FRANalysis:ENABle

 (see page 1292)

Command Syntax

```
:FRANalysis:ENABle <setting>

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

The :FRANalysis:ENABle command turns the Frequency Response Analysis (FRA) feature on or off.

Query Syntax

```
:FRANalysis:ENABle?
```

The :FRANalysis:ENABle? query returns a flag indicating whether the Frequency Response Analysis (FRA) feature is on or off.

Return Format

```
<setting><NL>

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

See Also

- ":FRANalysis:DATA?" on page 403
- ":FRANalysis:FREQuency:STARt" on page 407
- ":FRANalysis:FREQuency:STOP" on page 408
- ":FRANalysis:RUN" on page 410
- ":FRANalysis:SOURce:INPut" on page 411
- ":FRANalysis:SOURce:OUTPut" on page 412
- ":FRANalysis:WGEN:LOAD" on page 414
- ":FRANalysis:WGEN:VOLTage" on page 415

:FRANalysis:FREQuency:MODE

 (see page 1292)

Command Syntax

```
:FRANalysis:FREQuency:MODE <setting>

<setting> ::= {SWEep | SINGle}
```

The :FRANalysis:FREQuency:MODE command lets you select between the normal swept frequency response analysis or analysis at a single frequency, which can be useful when debugging.

Query Syntax

```
:FRANalysis:FREQuency:MODE?
```

The :FRANalysis:FREQuency:MODE? query returns the frequency mode setting.

Return Format

```
<setting><NL>

<setting> ::= {SWEep | SINGle}
```

See Also

- ":FRANalysis:RUN" on page 410
- ":FRANalysis:FREQuency:SINGle" on page 406
- ":FRANalysis:FREQuency:STARt" on page 407
- ":FRANalysis:FREQuency:STOP" on page 408
- ":FRANalysis:PPDecade" on page 409
- ":FRANalysis:WGEN:VOLTage:PROFile" on page 416

:FRANalysis:FREQuency:SINGle

(see page 1292)

Command Syntax

```
:FRANalysis:FREQuency:SINGle <value>[suffix]

<value> ::= {20 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000
   | 2000000}

[suffix] ::= {Hz | kHz| MHz}
```

The :FRANalysis:FREQuency:SINGle command command sets the single frequency value. The frequency response analysis is displayed on a log scale Bode plot, so you can select from decade values in addition to the minimum frequency of 20 Hz.

Query Syntax

```
:FRANalysis:FREQuency:SINGle?
```

The :FRANalysis:FREQuency:SINGle? query returns the single frequency setting.

Return Format

```
<value><NL>

<value> ::= {20 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000
   | 2000000}
```

See Also

- ":FRANalysis:DATA?" on page 403
- ":FRANalysis:ENABle" on page 404
- ":FRANalysis:PPDecade" on page 409
- ":FRANalysis:FREQuency:MODE" on page 405
- ":FRANalysis:RUN" on page 410
- ":FRANalysis:SOURce:INPut" on page 411
- ":FRANalysis:SOURce:OUTPut" on page 412
- ":FRANalysis:WGEN:LOAD" on page 414
- ":FRANalysis:WGEN:VOLTage" on page 415
- ":FRANalysis:WGEN:VOLTage:PROFile" on page 416

:FRANalysis:FREQuency:STARt

(see page 1292)

Command Syntax

```
:FRANalysis:FREQuency:STARt <value>[suffix]

<value> ::= {20 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000}

[suffix] ::= {Hz | kHz| MHz}
```

The :FRANalysis:FREQuency:STARt command command sets the frequency sweep start value. The frequency response analysis is displayed on a log scale Bode plot, so you can select from decade values in addition to the minimum frequency of 20 Hz.

Query Syntax

```
:FRANalysis:FREQuency:STARt?
```

The :FRANalysis:FREQuency:STARt? query returns the frequency sweep start setting.

Return Format

```
<value><NL>

<value> ::= {20 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000}
```

See Also

- ":FRANalysis:DATA?" on page 403
- ":FRANalysis:ENABle" on page 404
- ":FRANalysis:FREQuency:STOP" on page 408
- ":FRANalysis:PPDecade" on page 409
- ":FRANalysis:FREQuency:MODE" on page 405
- ":FRANalysis:RUN" on page 410
- ":FRANalysis:SOURce:INPut" on page 411
- ":FRANalysis:SOURce:OUTPut" on page 412
- ":FRANalysis:WGEN:LOAD" on page 414
- ":FRANalysis:WGEN:VOLTage" on page 415
- ":FRANalysis:WGEN:VOLTage:PROFile" on page 416

:FRANalysis:FREQuency:STOP

(see page 1292)

Command Syntax

```
:FRANalysis:FREQuency:STOP <value>[suffix]

<value> ::= {100 | 1000 | 10000 | 100000 | 1000000 | 10000000 | 20000000
}

[suffix] ::= {Hz | kHz| MHz}
```

The :FRANalysis:FREQuency:STOP command sets the frequency sweep stop value. The frequency response analysis is displayed on a log scale Bode plot, so you can select from decade values in addition to the maximum frequency of 20 MHz.

Query Syntax

```
:FRANalysis:FREQuency:STOP?
```

The :FRANalysis:FREQuency:STOP? query returns the frequency sweep stop setting.

Return Format

```
<value><NL>

<value> ::= {100 | 1000 | 10000 | 100000 | 1000000 | 10000000 | 20000000
}
```

See Also

- ":FRANalysis:DATA?" on page 403
- ":FRANalysis:ENABle" on page 404
- ":FRANalysis:FREQuency:STARt" on page 407
- ":FRANalysis:PPDecade" on page 409
- ":FRANalysis:FREQuency:MODE" on page 405
- ":FRANalysis:RUN" on page 410
- ":FRANalysis:SOURce:INPut" on page 411
- ":FRANalysis:SOURce:OUTPut" on page 412
- ":FRANalysis:WGEN:LOAD" on page 414
- ":FRANalysis:WGEN:VOLTage" on page 415
- ":FRANalysis:WGEN:VOLTage:PROFile" on page 416

:FRANalysis:PPDecade

(see page 1292)

Command Syntax

```
:FRANalysis:PPDecade <value>

<value> ::= {10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100}
```

The :FRANalysis:PPDecade command specifies the number of points per decade in the frequency response analysis.

Query Syntax

```
:FRANalysis:PPDecade?
```

The :FRANalysis:PPDecade? query returns the points per decade setting.

Return Format

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

See Also

- ":FRANalysis:FREQuency:STARt" on page 407
- ":FRANalysis:FREQuency:STOP" on page 408
- ":FRANalysis:WGEN:VOLTage:PROFile" on page 416

:FRANalysis:RUN

N (see page 1292)

Command Syntax `:FRANalysis:RUN`

The :FRANalysis:RUN command performs the Frequency Response Analysis. This analysis controls the built-in waveform generator to sweep a sine wave across a range of frequencies while measuring the input to and output from a device under test (DUT). At each frequency, gain (A) and phase are measured and plotted on a Bode frequency response chart.

The :FRANalysis:APPLy command is a valid compatible alias for the :FRANalysis:RUN command.

When the frequency response analysis completes, you can use the :FRANalysis:DATA? query to get four comma-separated columns of data for each step in the sweep: Frequency (Hz), Amplitude (Vpp), Gain (dB), and Phase (°).

You can use the :FRANalysis:TRACe command to specify whether to include gain, phase, both gain and phase, or neither in the frequency response analysis results.

It takes some time for the frequency sweep analysis to complete. You can query bit 0 of the Standard Event Status Register (*ESR?) to find out when the analysis is complete.

See Also

- ":FRANalysis:DATA?" on page 403
- ":FRANalysis:ENABle" on page 404
- ":FRANalysis:FREQuency:MODE" on page 405
- ":FRANalysis:FREQuency:SINGle" on page 406
- ":FRANalysis:FREQuency:STARt" on page 407
- ":FRANalysis:FREQuency:STOP" on page 408
- ":FRANalysis:SOURce:INPut" on page 411
- ":FRANalysis:SOURce:OUTPut" on page 412
- ":FRANalysis:TRACe" on page 413
- ":FRANalysis:WGEN:LOAD" on page 414
- ":FRANalysis:WGEN:VOLTage" on page 415
- "*ESR? (Standard Event Status Register)" on page 187

:FRANalysis:SOURce:INPut

 (see page 1292)

Command Syntax

```
:FRANalysis:SOURce:INPut <source>

<source> ::= CHANnel<n>

<n> ::= 1 to (# analog channels) in NR1 format
```

The :FRANalysis:SOURce:INPut command specifies the analog input channel that is probing the input voltage to the device under test (DUT) in the frequency response analysis.

Query Syntax

```
:FRANalysis:SOURce:INPut?
```

The :FRANalysis:SOURce:INPut? query returns the currently selected channel probing the input voltage.

Return Format

```
<source><NL>

<source> ::= CHAN<n>
```

See Also

- ":FRANalysis:DATA?" on page 403
- ":FRANalysis:ENABle" on page 404
- ":FRANalysis:FREQuency:STARt" on page 407
- ":FRANalysis:FREQuency:STOP" on page 408
- ":FRANalysis:RUN" on page 410
- ":FRANalysis:SOURce:OUTPut" on page 412
- ":FRANalysis:WGEN:LOAD" on page 414
- ":FRANalysis:WGEN:VOLTage" on page 415

:FRANalysis:SOURce:OUTPut

(see page 1292)

Command Syntax

```
:FRANalysis:SOURce:OUTPut <source>

<source> ::= CHANnel<n>

<n> ::= 1 to (# analog channels) in NR1 format
```

The :FRANalysis:SOURce:OUTPut command specifies the analog input channel that is probing the output voltage from the device under test (DUT) in the frequency response analysis.

Query Syntax

```
:FRANalysis:SOURce:OUTPut?
```

The :FRANalysis:SOURce:OUTPut? query returns the currently selected channel probing the output voltage.

Return Format

```
<source><NL>

<source> ::= CHAN<n>
```

See Also

- ":FRANalysis:DATA?" on page 403
- ":FRANalysis:ENABle" on page 404
- ":FRANalysis:FREQuency:STARt" on page 407
- ":FRANalysis:FREQuency:STOP" on page 408
- ":FRANalysis:RUN" on page 410
- ":FRANalysis:SOURce:INPut" on page 411
- ":FRANalysis:WGEN:LOAD" on page 414
- ":FRANalysis:WGEN:VOLTage" on page 415

:FRANalysis:TRACe

(see page 1292)

Command Syntax

```
:FRANalysis:TRACe <selection>

<selection> ::= {NONE | ALL | GAIN | PHASe}[,{GAIN | PHASe}]
```

The :FRANalysis:TRACe command specifies whether to include gain, phase, both gain and phase, or neither in the frequency response analysis results.

NOTE This command affects the oscilloscope's front panel graphical user interface (plot and table) as well as when saving analysis data.

Query Syntax

```
:FRANalysis:TRACe?
```

The :FRANalysis:TRACe? query returns a comma-separated list of the types of data that are currently included in the frequency response analysis results, or "NONE" if neither gain nor phase data is included.

Return Format

```
<selection_list><NL>

<selection_list> ::= {"NONE" | "GAIN" | "PHASe" | "GAIN,PHASe"}
```

See Also

- ":FRANalysis:RUN" on page 410
- ":FRANalysis:DATA?" on page 403

:FRANalysis:WGEN:LOAD

 (see page 1292)

Command Syntax

```
:FRANalysis:WGEN:LOAD <impedance>

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

The :FRANalysis:WGEN:LOAD command selects the expected output load impedance.

The output impedance of the Gen Out BNC is fixed at 50 ohms. However, the output load selection lets the waveform generator display the correct amplitude and offset levels for the expected output load.

If the actual load impedance is different than the selected value, the displayed amplitude and offset levels will be incorrect.

Query Syntax

```
:FRANalysis:WGEN:LOAD?
```

The :FRANalysis:WGEN:LOAD? query returns the current expected output load impedance.

Return Format

```
<impedance><NL>

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

See Also

- ":FRANalysis:DATA?" on page 403
- ":FRANalysis:ENABle" on page 404
- ":FRANalysis:FREQuency:STARt" on page 407
- ":FRANalysis:FREQuency:STOP" on page 408
- ":FRANalysis:RUN" on page 410
- ":FRANalysis:SOURce:INPut" on page 411
- ":FRANalysis:SOURce:OUTPut" on page 412
- ":FRANalysis:WGEN:VOLTage" on page 415

:FRANalysis:WGEN:VOLTage

(see page 1292)

Command Syntax

```
:FRANalysis:WGEN:VOLTage <amplitude>,[<range>]

<amplitude> ::= amplitude in volts in NR3 format

<range> ::= {F20HZ | F100HZ | F1KHZ | F10KHZ | F100KHZ | F1MHZ
   | F10MHZ | F20MHZ}
```

The :FRANalysis:WGEN:VOLTage command specifies the waveform generator's output sine wave amplitude.

Use the :WGEN:VOLTage:OFFSet command to specify the offset voltage or DC level.

You can also specify the amplitude and offset indirectly using the :WGEN:VOLTage:HIGH and :WGEN:VOLTage:LOW commands. For example, an amplitude of 5 V and an offset of 1 V is the same as a high-level voltage of 4 V and a low-level voltage of -1 V.

When the amplitude profile setting is on (:FRANalysis:WGEN:VOLTage:PROFile ON), the <range> option specifies the initial ramp amplitude at the frequency setting. Amplitudes ramp between the settings specified for individual frequencies.

Query Syntax

```
:FRANalysis:WGEN:VOLTage? [<range>]
```

The :FRANalysis:WGEN:VOLTage? query returns the currently specified waveform generator amplitude.

When the amplitude profile setting is on (:FRANalysis:WGEN:VOLTage:PROFile ON), the <range> option specifies the frequency whose initial ramp amplitude is returned.

Return Format

```
<amplitude><NL>

<amplitude> ::= amplitude in volts in NR3 format
```

See Also

- ":FRANalysis:WGEN:VOLTage:PROFile" on page 416
- ":FRANalysis:DATA?" on page 403
- ":FRANalysis:ENABle" on page 404
- ":FRANalysis:FREQuency:STARt" on page 407
- ":FRANalysis:FREQuency:STOP" on page 408
- ":FRANalysis:PPDecade" on page 409
- ":FRANalysis:RUN" on page 410
- ":FRANalysis:SOURce:INPut" on page 411
- ":FRANalysis:SOURce:OUTPut" on page 412
- ":FRANalysis:WGEN:LOAD" on page 414

:FRANalysis:WGEN:VOLTage:PROFile

(see page 1292)

Command Syntax `:FRANalysis:WGEN:VOLTage:PROFile {{0 | OFF} | {1 | ON}}`

The :FRANalysis:WGEN:VOLTage:PROFile command enables or disables the ability to specify amplitude ramping within different decades.

Query Syntax `:FRANalysis:WGEN:VOLTage:PROFile?`

The :FRANalysis:WGEN:VOLTage:PROFile? query returns the voltage profile setting.

Return Format `<setting><NL>`

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

See Also

- ":FRANalysis:WGEN:VOLTage" on page 415
- ":FRANalysis:PPDecade" on page 409
- ":FRANalysis:FREQuency:STARt" on page 407
- ":FRANalysis:FREQuency:STOP" on page 408

19 :FUNCtion<m> Commands

Control math functions in the oscilloscope. See "Introduction to :FUNCtion<m> Commands" on page 422.

Table 84 :FUNCtion<m> Commands Summary

Command	Query	Options and Query Returns
:FUNCtion<m>:AVERage:COUNt <count> (see page 424)	:FUNCtion<m>:AVERage:COUNt? (see page 424)	<count> ::= an integer from 2 to 65536 in NR1 format <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:BUS:CLOCk <source> (see page 425)	:FUNCtion<m>:BUS:CLOCk? (see page 425)	<source> ::= {CHANnel<n> | DIGital<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:BUS:SLOPe <slope> (see page 426)	:FUNCtion<m>:BUS:SLOPe? (see page 426)	<slope> ::= {NEGative | POSitive | EITHer} <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:BUS:YINCrement <value> (see page 427)	:FUNCtion<m>:BUS:YINCrement? (see page 427)	<value> ::= value per bus code, in NR3 format <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:BUS:YORigin <value> (see page 428)	:FUNCtion<m>:BUS:YORigin? (see page 428)	<value> ::= value at bus code = 0, in NR3 format <m> ::= 1 to (# math functions) in NR1 format

Table 84 :FUNCtion<m> Commands Summary (continued)

Command	Query	Options and Query Returns
:FUNCtion<m>:BUS:YUNits <units> (see page 429)	:FUNCtion<m>:BUS:YUNits? (see page 429)	<units> ::= {VOLT | AMPere | NONE} <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:CLEar (see page 430)	n/a	n/a
:FUNCtion<m>:DISPlay {{0 | OFF} | {1 | ON}} (see page 431)	:FUNCtion<m>:DISPlay? (see page 431)	{0 | 1} <m> ::= 1 to (# math functions) in NR1 format
n/a	:FUNCtion<m>[:FFT]:BSIZe? (see page 432)	<bin_size> ::= Hz in NR3 format <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>[:FFT]:CENTer <frequency> (see page 433)	:FUNCtion<m>[:FFT]:CENTer? (see page 433)	<frequency> ::= the current center frequency in NR3 format. The range of legal values is from -25 GHz to 25 GHz. <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>[:FFT]:DETection:POINts <number_of_buckets> (see page 434)	:FUNCtion<m>[:FFT]:DETection:POINts? (see page 434)	<number_of_buckets> ::= an integer in NR1 format <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>[:FFT]:DETection:TYPE <type> (see page 435)	:FUNCtion<m>[:FFT]:DETection:TYPE? (see page 435)	<type> ::= {OFF | SAMPle | PPOSitive | PNEGative | NORMal | AVERage} <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>[:FFT]:FREQuency:STARt <frequency> (see page 436)	:FUNCtion<m>[:FFT]:FREQuency:STARt? (see page 436)	<frequency> ::= the start frequency in NR3 format. <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>[:FFT]:FREQuency:STOP <frequency> (see page 437)	:FUNCtion<m>[:FFT]:FREQuency:STOP? (see page 437)	<frequency> ::= the stop frequency in NR3 format. <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>[:FFT]:GATE <gating> (see page 438)	:FUNCtion<m>[:FFT]:GATE? (see page 438)	<gating> ::= {NONE | ZOOM} <m> ::= 1-4 in NR1 format

Table 84 :FUNCtion<m> Commands Summary (continued)

Command	Query	Options and Query Returns
:FUNCtion<m>[:FFT]:PHASe:REFerence <ref_point> (see page 439)	:FUNCtion<m>[:FFT]:PHASe:REFerence? (see page 439)	<ref_point> ::= {TRIGger | DISPlay} <m> ::= 1-4 in NR1 format
n/a	:FUNCtion<m>[:FFT]:RBWidth? (see page 440)	<resolution_bw> ::= Hz in NR3 format <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>[:FFT]:READout<n> <readout_type> (see page 441)	:FUNCtion<m>[:FFT]:READout<n>? (see page 441)	<readout_type> ::= {SRATe | BSIZe | RBWidth} <m> ::= 1 to (# math functions) in NR1 format <n> ::= 1-2 in NR1 format, 2 is for dedicated FFT function
:FUNCtion<m>[:FFT]:SPAN (see page 442)	:FUNCtion<m>[:FFT]:SPAN? (see page 442)	::= the current frequency span in NR3 format. Legal values are 1 Hz to 100 GHz. <m> ::= 1 to (# math functions) in NR1 format
n/a	:FUNCtion<m>[:FFT]:SRATe? (see page 443)	<sample_rate> ::= Hz in NR3 format <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>[:FFT]:VTYPe <units> (see page 444)	:FUNCtion<m>[:FFT]:VTYPe? (see page 444)	<units> ::= {DECibel | VRMS} for the FFT (magnitude) operation <units> ::= {DEGRees | RADians} for the FFTPhase operation <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>[:FFT]:WINDow <window> (see page 445)	:FUNCtion<m>[:FFT]:WINDow? (see page 445)	<window> ::= {RECTangular | HANNing | FLATtop | BHARris | BARTlett} <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:FREQuency:BANDpass:CENTer <center_freq> (see page 446)	:FUNCtion<m>:FREQuency:BANDpass:CENTer? (see page 446)	<center_freq> ::= center frequency of band-pass filter in NR3 format

Table 84 :FUNCtion<m> Commands Summary (continued)

Command	Query	Options and Query Returns
:FUNCtion<m>:FREQuency:BANDpass:WIDTh <freq_width> (see page 447)	:FUNCtion<m>:FREQuency:BANDpass:WIDth? (see page 447)	<freq_width> ::= frequency width of band-pass filter in NR3 format
:FUNCtion<m>:FREQuency:HIGHpass <3dB_freq> (see page 448)	:FUNCtion<m>:FREQuency:HIGHpass? (see page 448)	<3dB_freq> ::= 3dB cutoff frequency value in NR3 format <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:FREQuency:LOWPass <3dB_freq> (see page 449)	:FUNCtion<m>:FREQuency:LOWPass? (see page 449)	<3dB_freq> ::= 3dB cutoff frequency value in NR3 format <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:INTegrate:IICondition {{0 | OFF} | {1 | ON}} (see page 450)	:FUNCtion<m>:INTegrate:IICondition? (see page 450)	{0 | 1} <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:INTegrate:IOFFset <input_offset> (see page 451)	:FUNCtion<m>:INTegrate:IOFFset? (see page 451)	<input_offset> ::= DC offset correction in NR3 format. <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:LABel <string> (see page 452)	:FUNCtion<m>:LABel? (see page 452)	<string> ::= quoted ASCII string
:FUNCtion<m>:LINear:GAIN <value> (see page 453)	:FUNCtion<m>:LINear:GAIN? (see page 453)	<value> ::= 'A' in Ax + B, value in NR3 format <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:LINear:OFFSet <value> (see page 454)	:FUNCtion<m>:LINear:OFFSet? (see page 454)	<value> ::= 'B' in Ax + B, value in NR3 format <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:OFFSet <offset> (see page 455)	:FUNCtion<m>:OFFSet? (see page 455)	<offset> ::= the value at center screen in NR3 format. The range of legal values is +/-10 times the current sensitivity of the selected function. <m> ::= 1 to (# math functions) in NR1 format

Table 84 :FUNCtion<m> Commands Summary (continued)

Command	Query	Options and Query Returns
:FUNCtion<m>:OPERation <operation> (see page 456)	:FUNCtion<m>:OPERation? (see page 458)	<operation> ::= {ADD | SUBTract | MULTiply | DIVide | INTegrate | DIFF | FFT | FFTPhase | SQRT | MAGNify | ABSolute | SQUare | LN | LOG | EXP | TEN | LOWPass | HIGHpass | BANDpass | AVERage | LINear | MAXimum | MINimum | MAXHold | MINHold | TRENd} <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:RANGe <range> (see page 460)	:FUNCtion<m>:RANGe? (see page 460)	<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 DIFF function is 80E-3 to 8.0E12 (depends on current sweep speed). The range for the FFT function is 8 to 800 dBV. <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:REFerence <level> (see page 461)	:FUNCtion<m>:REFerence? (see page 461)	<level> ::= the value at center screen in NR3 format. The range of legal values is +/-10 times the current sensitivity of the selected function. <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:SCALe <scale value>[<suffix>] (see page 462)	:FUNCtion<m>:SCALe? (see page 462)	<scale value> ::= integer in NR1 format <suffix> ::= {V | dB} <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:SMOoth:POINts <points> (see page 463)	:FUNCtion<m>:SMOoth:POINts? (see page 463)	<points> ::= odd integer in NR1 format

Table 84 :FUNCtion<m> Commands Summary (continued)

Command	Query	Options and Query Returns
:FUNCtion<m>:SOURce1 <source> (see page 464)	:FUNCtion<m>:SOURce1? (see page 464)	<source> ::= {CHANnel<n> | FUNCtion<c> | MATH<c> | WMEMory<r> | BUS<b>} <n> ::= 1 to (# analog channels) in NR1 format <c> ::= {1 | 2 | 3}, must be lower than <m> <r> ::= 1 to (# ref waveforms) in NR1 format <b> ::= {1 | 2} <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:SOURce2 <source> (see page 466)	:FUNCtion<m>:SOURce2? (see page 466)	<source> ::= {CHANnel<n> | WMEMory<r> | NONE} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:TRENd:NMEasurement MEAS<n> (see page 467)	:FUNCtion<m>:TRENd:NMEasurement? (see page 467)	<n> ::= # of installed measurement, from 1 to 8 <m> ::= 1 to (# math functions) in NR1 format

Introduction to :FUNCtion<m> Commands

The FUNCtion subsystem controls the math functions in the oscilloscope. Two math functions are available — the <m> in :FUNCtion<m> can be from 1 to 2. However, a dedicated FFT function is also available, see Chapter 17, ":FFT Commands," starting on page 375.

The math function operator, transform, filter, or visualization is selected using the :FUNCtion<m>:OPERation command. Depending on the selected operation, there may be other commands for specifying options for that operation. See ":FUNCtion<m>:OPERation" on page 456.

The SOURce1, DISPlay, RANGe, and OFFSet (or REFerence) commands apply to any function.

Reporting the Setup

Use :FUNCtion<m>? to query setup information for the FUNCtion subsystem.

Return Format

The following is a sample response from the :FUNCtion1? query. In this case, the query was issued following a *RST command.

```
:FUNC1:OPER LIN;DISP 0;SOUR1 CHAN1;RANG +8.00E+00;OFFS +0.0E+00;
LIN:GAIN +1.00E+00;OFFS +0.0E+00
```

:FUNCtion<m>:AVERage:COUNt

(see page 1292)

Command Syntax

```
:FUNCtion<m>:AVERage:COUNt <count>

<count> ::= an integer from 2 to 65536 in NR1 format

<m> ::= 1 to (# math functions) in NR1 format
```

The :FUNCtion<m>:AVERage:COUNt command sets the number of waveforms to be averaged together.

The number of averages can be set from 2 to 65536 in increments of powers of 2.

Increasing the number of averages will increase resolution and reduce noise.

Query Syntax

```
:FUNCtion<m>:AVERage:COUNt?
```

The :FUNCtion<m>:AVERage:COUNt? query returns the number of waveforms to be averaged together.

Return Format

```
<count><NL>

<count> ::= an integer from 2 to 65536 in NR1 format
```

See Also

- ":FUNCtion<m>:OPERation" on page 456

:FUNCtion<m>:BUS:CLOCk

(see page 1292)

Command Syntax

```
:FUNCtion<m>:BUS:CLOCk <source>

<m> ::= 1 to (# math functions) in NR1 format

<source> ::= {DIGital<d>}

<d> ::= 0 to (# digital channels - 1) in NR1 format
```

The :FUNCtion<m>:BUS:CLOCk command selects the clock signal source for the Chart Logic Bus State operation.

Query Syntax

```
:FUNCtion<m>:BUS:CLOCk?
```

The :FUNCtion<m>:BUS:CLOCk query returns the source selected for the clock signal.

Return Format

```
<source><NL>

<source> ::= {DIGital<d>}

<d> ::= 0 to (# digital channels - 1) in NR1 format
```

See Also

- ":FUNCtion<m>:OPERation" on page 456

:FUNCtion<m>:BUS:SLOPe

 (see page 1292)

Command Syntax

```
:FUNCtion<m>:BUS:SLOPe <slope>

<m> ::= 1 to (# math functions) in NR1 format

<slope> ::= {NEGative | POSitive | EITHer}
```

The :FUNCtion<m>:BUS:SLOPe command specifies the clock signal edge for the Chart Logic Bus State operation.

Query Syntax

```
:FUNCtion<m>:BUS:SLOPe?
```

The :FUNCtion<m>:BUS:SLOPe query returns the clock edge setting.

Return Format

```
<slope><NL>

<slope> ::= {NEGative | POSitive | EITHer}
```

See Also

- ":FUNCtion<m>:OPERation" on page 456

:FUNCtion<m>:BUS:YINCrement

(see page 1292)

Command Syntax

```
:FUNCtion<m>:BUS:YINCrement <value>

<m> ::= 1 to (# math functions) in NR1 format

<value> ::= value per bus code, in NR3 format
```

The :FUNCtion<m>:BUS:YINCrement command specifies the value associated with each increment in Chart Logic Bus data.

Query Syntax

```
:FUNCtion<m>:BUS:YINCrement?
```

The :FUNCtion<m>:BUS:YINCrement query returns the value associated with each increment in Chart Logic Bus data.

Return Format

```
<value><NL>

<value> ::= value per bus code, in NR3 format
```

See Also

- ":FUNCtion<m>:OPERation" on page 456

:FUNCtion<m>:BUS:YORigin

 (see page 1292)

Command Syntax

```
:FUNCtion<m>:BUS:YORigin <value>

<m> ::= 1 to (# math functions) in NR1 format

<value> ::= value at bus code = 0, in NR3 format
```

The :FUNCtion<m>:BUS:YORigin command specifies the value associated with Chart Logic Bus data equal to zero.

Query Syntax

```
:FUNCtion<m>:BUS:YORigin?
```

The :FUNCtion<m>:BUS:YORigin query returns the value for associated with data equal to zero.

Return Format

```
<value><NL>

<value> ::= value at bus code = 0, in NR3 format
```

See Also

- ":FUNCtion<m>:OPERation" on page 456

:FUNCtion<m>:BUS:YUNits

(see page 1292)

Command Syntax

```
:FUNCtion<m>:BUS:YUNits <units>

<m> ::= 1 to (# math functions) in NR1 format

<units> ::= {VOLT | AMPere | NONE}
```

The :FUNCtion<m>:BUS:YUNits command specifies the vertical units for the Chart Logic Bus operations.

Query Syntax

```
:FUNCtion<m>:BUS:YUNits?
```

The :FUNCtion<m>:BUS:YUNits query returns the Chart Logic Bus vertical units.

Return Format

```
<units><NL>

<units> ::= {VOLT | AMP | NONE}
```

See Also

- ":FUNCtion<m>:OPERation" on page 456

:FUNCtion<m>:CLEar

(see page 1292)

Command Syntax `:FUNCtion<m>:CLEar`

When the :FUNCtion<m>:OPERation is AVERage, MAXHold, or MINHold, the :FUNCtion<m>:CLEar command clears the number of evaluated waveforms.

See Also

- ":FUNCtion<m>:AVERage:COUNt" on page 424

:FUNCtion<m>:DISPlay

(see page 1292)

Command Syntax

```
:FUNCtion<m>:DISPlay <display>

<m> ::= 1 to (# math functions) in NR1 format

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

The :FUNCtion<m>:DISPlay command turns the display of the function on or off. When ON is selected, the function operates as specified by the other :FUNCtion<m> commands.

NOTE

Automatic vertical scaling of the math function waveform occurs when the function display is turned from off to on. If you wait for the operation to complete (with an *OPC? query for example), then query the math function's scale, you can see the vertical scaling that was automatically determined.

One math function can be displayed at a time. If one math function is on and then another math function is turned on, the first math function will be turned off.

When math functions are off, they are still calculated so that they can be cascaded, that is, used as a source for a higher math function.

Query Syntax

```
:FUNCtion<m>:DISPlay?
```

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

Return Format

```
<display><NL>

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

See Also

- "Introduction to :FUNCtion<m> Commands" on page 422
- "*OPC (Operation Complete)" on page 191
- ":FUNCtion<m>:SCALe" on page 462
- ":VIEW" on page 227
- ":BLANk" on page 217
- ":STATus?" on page 224

:FUNCtion<m>[:FFT]:BSIZe?

 (see page 1292)

Query Syntax

```
:FUNCtion<m>[:FFT]:BSIZe?

<m> ::= 1-4 in NR1 format
```

The :FUNCtion<m>[:FFT]:BSIZe? query returns the Bin Size setting for the FFT.

Return Format

```
<bin_size><NL>

<bin_size> ::= Hz in NR3 format
```

See Also

- ":FUNCtion<m>[:FFT]:READout<n>" on page 441

:FUNCtion<m>[:FFT]:CENTer

(see page 1292)

Command Syntax

```
:FUNCtion<m>[:FFT]:CENTer <frequency>

<m> ::= 1 to (# math functions) in NR1 format

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

The :FUNCtion<m>[:FFT]:CENTer command sets the center frequency when FFT (Fast Fourier Transform) is selected.

Query Syntax

```
:FUNCtion<m>[:FFT]:CENTer?
```

The :FUNCtion<m>[:FFT]: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 -25 GHz to 25 GHz.
```

NOTE

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

See Also

- "Introduction to :FUNCtion<m> Commands" on page 422
- ":FUNCtion<m>[:FFT]:SPAN" on page 442
- ":TIMebase[:MAIN]:RANGe" on page 1068
- ":TIMebase[:MAIN]:SCALe" on page 1069

:FUNCtion<m>[:FFT]:DETection:POINts

(see page 1292)

Command Syntax

```
:FUNCtion<m>[:FFT]:DETection:POINts <number_of_buckets>

<number_of_buckets> ::= an integer from 640 to 64K in NR1 format

<m> ::= 1-4 in NR1 format
```

The :FUNCtion<m>[:FFT]:DETection:POINts command specifies the maximum number of points that the FFT detector should decimate to. This is also the number of buckets that sampled FFT points are grouped into before the selected detection type reduction (decimation) is applied.

The minimum number of points is 640.

The maximum number of points is the analysis record limit. The analysis record defaults to 64K points but its depth can be increased at the expense of waveform update rate (see :SYSTem:PRECision:LENGth).

Query Syntax

```
:FUNCtion<m>[:FFT]:DETection:POINts?
```

The :FUNCtion<m>[:FFT]:DETection:POINts? query returns the FFT detector points setting.

Return Format

```
<number_of_buckets><NL>

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

See Also

- ":FUNCtion<m>[:FFT]:DETection:TYPE" on page 435
- ":SYSTem:PRECision:LENGth" on page 1043

:FUNCtion<m>[:FFT]:DETection:TYPE

N (see page 1292)

Command Syntax

```
:FUNCtion<m>[:FFT]:DETection:TYPE <type>

<type> ::= {OFF | SAMPle | PPOSitive | PNEGative | NORMal | AVERage}

<m> ::= 1-4 in NR1 format
```

The :FUNCtion<m>[:FFT]:DETection:TYPE command sets the FFT detector decimation type.

Detectors give you a way of manipulating the acquired data to emphasize different features of the data. Detectors reduce (decimate) the number of FFT points to at most the number of specified detector points. In this reduction, sampled FFT points are bucketized, that is, split into a number of groups that equals the specified number of detector points. Then, the points in each bucket are reduced to a single point according to the selected detection type. The detector types are:

- OFF – No detector is used.
- SAMPle – Takes the point nearest to the center of every bucket.
- PPOSitive (+ Peak) – Takes the most positive point in every bucket.
- PNEGative (- Peak) – Takes the most negative point in every bucket.
- AVERage – Takes the average of all points in every bucket.
- NORMal – Implements a rosenfell algorithm. This method to picks either the minimum or maximum sample in every bucket depending on whether the data is monotonically increasing, decreasing, or varying. For details, see the Spectrum Analysis Basics application note at www.keysight.com.

When detectors are used, the FFT's output is decimated, and any analysis is performed on the reduced or detected data set.

Query Syntax

```
:FUNCtion<m>[:FFT]:DETection:TYPE?
```

The :FUNCtion<m>[:FFT]:DETection:TYPE? query returns the FFT detector type setting

Return Format

```
<type><NL>

<type> ::= {OFF | SAMPle | PPOSitive | PNEGative | NORMal | AVERage}
```

See Also

- ":FUNCtion<m>[:FFT]:DETection:POINts" on page 434

:FUNCtion<m>[:FFT]:FREQuency:STARt

N (see page 1292)

Command Syntax

```
:FUNCtion<m>[:FFT]:FREQuency:STARt <frequency>

<m> ::= 1 to (# math functions) in NR1 format

<frequency> ::= the start frequency in NR3 format.
```

The :FUNCtion<m>[:FFT]:FREQuency:STARt command sets the start frequency in the FFT (Fast Fourier Transform) math function's displayed range.

The FFT (Fast Fourier Transform) math function's displayed range can also be set with the :FUNCtion<m>[:FFT]:CENTer and :FUNCtion<m>[:FFT]:SPAN commands.

Query Syntax

```
:FUNCtion<m>[:FFT]:FREQuency:STARt?
```

The :FUNCtion<m>[:FFT]:FREQuency:STARt? query returns the current start frequency in Hertz.

Return Format

```
<frequency><NL>

<frequency> ::= the start frequency in NR3 format.
```

See Also

- ":FUNCtion<m>[:FFT]:FREQuency:STOP" on page 437
- ":FUNCtion<m>[:FFT]:CENTer" on page 433
- ":FUNCtion<m>[:FFT]:SPAN" on page 442

:FUNCtion<m>[:FFT]:FREQuency:STOP

N (see page 1292)

Command Syntax

```
:FUNCtion<m>[:FFT]:FREQuency:STOP <frequency>

<m> ::= 1 to (# math functions) in NR1 format

<frequency> ::= the stop frequency in NR3 format.
```

The :FUNCtion<m>[:FFT]:FREQuency:STOP command sets the stop frequency in the FFT (Fast Fourier Transform) math function's displayed range.

The FFT (Fast Fourier Transform) math function's displayed range can also be set with the :FUNCtion<m>[:FFT]:CENTer and :FUNCtion<m>[:FFT]:SPAN commands.

Query Syntax

```
:FUNCtion<m>[:FFT]:FREQuency:STOP?
```

The :FUNCtion<m>[:FFT]:FREQuency:STOP? query returns returns the current stop frequency in Hertz.

Return Format

```
<frequency><NL>

<frequency> ::= the stop frequency in NR3 format.
```

See Also

- ":FUNCtion<m>[:FFT]:FREQuency:STARt" on page 436
- ":FUNCtion<m>[:FFT]:CENTer" on page 433
- ":FUNCtion<m>[:FFT]:SPAN" on page 442

:FUNCtion<m>[:FFT]:GATE

 (see page 1292)

Command Syntax

```
:FUNCtion<m>[:FFT]:GATE <gating>

<m> ::= 1-4 in NR1 format

<gating> ::= {NONE | ZOOM}
```

The :FUNCtion<m>[:FFT]:GATE command specifies whether the FFT is performed on the Main time base window (NONE) or the ZOOM window when the zoomed time base is displayed.

Query Syntax

```
:FUNCtion<m>[:FFT]:GATE?
```

The :FUNCtion<m>[:FFT]:GATE? query returns the gate setting.

Return Format

```
<gating><NL>

<gating> ::= {NONE | ZOOM}
```

See Also

- ":FUNCtion<m>[:FFT]:VTYPe" on page 444
- ":FUNCtion<m>[:FFT]:WINDow" on page 445
- "Introduction to :FUNCtion<m> Commands" on page 422
- ":FUNCtion<m>:OPERation" on page 456

:FUNCtion<m>[:FFT]:PHASe:REFerence

 (see page 1292)

Command Syntax

```
:FUNCtion<m>[:FFT]:PHASe:REFerence <ref_point>

<ref_point> ::= {TRIGger | DISPlay}

<m> ::= 1-4 in NR1 format
```

The :FUNCtion<m>[:FFT]:PHASe:REFerence command sets the reference point for calculating the FFT Phase function to either the trigger point or beginning of the displayed waveform.

Query Syntax

```
:FUNCtion<m>[:FFT]:PHASe:REFerence?
```

The :FUNCtion<m>[:FFT]:PHASe:REFerence? query returns the selected reference point.

Return Format

```
<ref_point><NL>

<ref_point> ::= {TRIGger | DISPlay}
```

See Also

- ":FUNCtion<m>:OPERation" on page 456
- "Introduction to :FUNCtion<m> Commands" on page 422

:FUNCtion<m>[:FFT]:RBWidth?

(see page 1292)

Query Syntax

```
:FUNCtion<m>[:FFT]:RBWidth?

<m> ::= 1-4 in NR1 format
```

The :FUNCtion<m>[:FFT]:RBWidth? query returns the Resolution Bandwidth setting for the FFT.

Return Format

```
<resolution_bw><NL>

<resolution_bw> ::= Hz in NR3 format
```

See Also

- ":FUNCtion<m>[:FFT]:READout<n>" on page 441

:FUNCtion<m>[:FFT]:READout<n>

N (see page 1292)

Command Syntax

```
:FUNCtion<m>[:FFT]:READout<n> <readout_type>

<readout_type> ::= {SRATe | BSIZe | RBWidth}

<m> ::= 1-4 in NR1 format

<n> ::= 1-2 in NR1 format, 2 is valid only on oscilloscopes that have
   the dedicated FFT function
```

The :FUNCtion<m>[:FFT]:READout<n> command selects from these types of readouts for the FFT:

- SRATe – Sample Rate
- BSIZe – Bin Size
- RBWidth – Resolution Bandwidth

Note that READout1 is used for both the dedicated FFT and the general-purpose math FFT function and READout2 is used only for oscilloscopes that have a dedicated FFT function (like the 3000T X-Series).

Query Syntax

```
:FUNCtion<m>[:FFT]:READout<n>?
```

The :FUNCtion<m>[:FFT]:READout<n>? query returns the readout selection.

Return Format

```
<readout_type><NL>

<readout_type> ::= {SRAT | BSIZ | RBW}
```

See Also

- ":FUNCtion<m>[:FFT]:BSIZe?" on page 432
- ":FUNCtion<m>[:FFT]:RBWidth?" on page 440
- ":FUNCtion<m>[:FFT]:SRATe?" on page 443

:FUNCtion<m>[:FFT]:SPAN

N (see page 1292)

Command Syntax

```
:FUNCtion<m>[:FFT]:SPAN <span>

<m> ::= 1 to (# math functions) in NR1 format

<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<m>[:FFT]:SPAN command sets the frequency span of the display (left graticule to right graticule) when FFT (Fast Fourier Transform) is selected.

Query Syntax

```
:FUNCtion<m>[:FFT]:SPAN?
```

The :FUNCtion<m>[:FFT]:SPAN? query returns the current frequency span in Hertz.

NOTE

After a *RST (Reset) or :AUToscale command, the values returned by the :FUNCtion<m>[:FFT]:CENTer? and :FUNCtion<m>:SPAN? queries depend on the current :TIMebase:RANGe value. Once you change either the :FUNCtion<m>[:FFT]:CENTer or :FUNCtion<m>: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<m> Commands" on page 422
- ":FUNCtion<m>[:FFT]:CENTer" on page 433
- ":TIMebase[:MAIN]:RANGe" on page 1068
- ":TIMebase[:MAIN]:SCALe" on page 1069

:FUNCtion<m>[:FFT]:SRATe?

 (see page 1292)

Query Syntax

```
:FUNCtion<m>[:FFT]:SRATe?

<m> ::= 1-4 in NR1 format
```

The :FUNCtion<m>[:FFT]:SRATe? query returns the Sample Rate setting for the FFT.

Return Format

```
<sample_rate><NL>

<sample_rate> ::= Hz in NR3 format
```

See Also

- ":FUNCtion<m>[:FFT]:READout<n>" on page 441

:FUNCtion<m>[:FFT]:VTYPe

N (see page 1292)

Command Syntax

```
:FUNCtion<m>[:FFT]:VTYPe <units>

<m> ::= 1 to (# math functions) in NR1 format

<units> ::= {DECibel | VRMS} for the FFT (magnitude) operation

<units> ::= {DEGRees | RADians} for the FFTPhase operation
```

The :FUNCtion<m>[:FFT]:VTYPe command specifies FFT vertical units.

For the FFT (Magnitude) operation units, DECibel equates to the user interface's Logarithmic selection, and VRMS equates to the user interface's Linear selection.

Query Syntax

```
:FUNCtion<m>[:FFT]:VTYPe?
```

The :FUNCtion<m>[:FFT]:VTYPe? query returns the current FFT vertical units.

Return Format

```
<units><NL>

<units> ::= {DEC | VRMS} for the FFT (magnitude) operation

<units> ::= {DEGR | RAD} for the FFTPhase operation
```

See Also

- ":FUNCtion<m>[:FFT]:GATE" on page 438
- "Introduction to :FUNCtion<m> Commands" on page 422
- ":FUNCtion<m>:OPERation" on page 456

:FUNCtion<m>[:FFT]:WINDow

N (see page 1292)

Command Syntax

```
:FUNCtion<m>[:FFT]:WINDow <window>

<m> ::= 1 to (# math functions) in NR1 format

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

The :FUNCtion<m>[:FFT]:WINDow command allows the selection of 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.

- RECTangular – useful for transient signals, and signals where there are an integral number of cycles in the time record.
- HANNing – 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.
- FLATtop – best for making accurate amplitude measurements of frequency peaks.
- BHARris (Blackman-Harris) – reduces time resolution compared to the rectangular window, but it improves the capacity to detect smaller impulses due to lower secondary lobes (provides minimal spectral leakage).
- BARTlett – (triangular, with end points at zero) window is similar to the Hanning window in that it is good for making accurate frequency measurements, but its higher and wider secondary lobes make it not quite as good for resolving frequencies that are close together.

Query Syntax

```
:FUNCtion<m>[:FFT]:WINDow?
```

The :FUNCtion<m>[:FFT]:WINDow? query returns the value of the window selected for the FFT function.

Return Format

```
<window><NL>

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

See Also

- ":FUNCtion<m>[:FFT]:GATE" on page 438
- "Introduction to :FUNCtion<m> Commands" on page 422

:FUNCtion<m>:FREQuency:BANDpass:CENTer

(see page 1292)

Command Syntax

```
:FUNCtion<m>:FREQuency:BANDpass:CENTer <center_freq>

<center_freq> ::= center frequency of band-pass filter in NR3 format
```

The :FUNCtion<m>:FREQuency:BANDpass:CENTer command sets the center frequency of the band-pass filter.

Query Syntax

```
:FUNCtion<m>:FREQuency:BANDpass:CENTer?
```

The :FUNCtion<m>:FREQuency:BANDpass:CENTer? query returns the band-pass filter's center frequency setting.

Return Format

```
<center_freq><NL>
```

See Also

- ":FUNCtion<m>:FREQuency:BANDpass:WIDTh" on page 447
- ":FUNCtion<m>:OPERation" on page 456

:FUNCtion<m>:FREQuency:BANDpass:WIDTh

(see page 1292)

Command Syntax

```
:FUNCtion<m>:FREQuency:BANDpass:WIDTh <freq_width>

<freq_width> ::= frequency width of band-pass filter in NR3 format
```

The :FUNCtion<m>:FREQuency:BANDpass:WIDTh command sets the frequency width of the band-pass filter. The width specifies the filter's -3 dB cutoff frequencies (center frequency minus half the width and center frequency plus half the width).

Query Syntax

```
:FUNCtion<m>:FREQuency:BANDpass:WIDTh?
```

The :FUNCtion<m>:FREQuency:BANDpass:WIDTh? query returns the band-pass filter's frequency width setting.

Return Format

```
<freq_width><NL>
```

See Also

- ":FUNCtion<m>:FREQuency:BANDpass:CENTer" on page 446
- ":FUNCtion<m>:OPERation" on page 456

:FUNCtion<m>:FREQuency:HIGHpass

 (see page 1292)

Command Syntax

```
:FUNCtion<m>:FREQuency:HIGHpass <3dB_freq>

<m> ::= 1 to (# math functions) in NR1 format

<3dB_freq> ::= -3dB cutoff frequency value in NR3 format
```

The :FUNCtion<m>:FREQuency:HIGHpass command sets the high-pass filter's -3 dB cutoff frequency.

The high-pass filter is a single-pole high pass filter.

Query Syntax

```
:FUNCtion<m>:FREQuency:HIGHpass?
```

The :FUNCtion<m>:FREQuency:HIGHpass query returns the high-pass filter's cutoff frequency.

Return Format

```
<3dB_freq><NL>

<3dB_freq> ::= -3dB cutoff frequency value in NR3 format
```

See Also

- ":FUNCtion<m>:OPERation" on page 456

:FUNCtion<m>:FREQuency:LOWPass

 (see page 1292)

Command Syntax

```
:FUNCtion<m>:FREQuency:LOWPass <3dB_freq>

<m> ::= 1 to (# math functions) in NR1 format

<3dB_freq> ::= -3dB cutoff frequency value in NR3 format
```

The :FUNCtion<m>:FREQuency:LOWPass command sets the low-pass filter's -3 dB cutoff frequency.

The low-pass filter is a 4th order Bessel-Thompson filter.

Query Syntax

```
:FUNCtion<m>:FREQuency:LOWPass?
```

The :FUNCtion<m>:FREQuency:LOWPass query returns the low-pass filter's cutoff frequency.

Return Format

```
<3dB_freq><NL>

<3dB_freq> ::= -3dB cutoff frequency value in NR3 format
```

See Also

- ":FUNCtion<m>:OPERation" on page 456

:FUNCtion<m>:INTegrate:IICondition

 (see page 1292)

Command Syntax

```
:FUNCtion<m>:INTegrate:IICondition {{0 | OFF} | {1 | ON}}

<m> ::= 1 to (# math functions) in NR1 format
```

The :FUNCtion<m>:INTegrate:IICondition command sets the Initial Condition of the Integrate math function waveform:

- ON – the integrate math waveform is vertically centered in the screen. In other words, the top and bottom of the math waveform are equal distances from the top and bottom of the screen.
- OFF – the integrate math waveform starts at the zero-level reference on the left side of the screen.

Query Syntax

```
:FUNCtion<m>:INTegrate:IICondition?
```

The :FUNCtion<m>:INTegrate:IICondition? query returns Initial Condition setting for the Integrate math function.

Return Format

```
<setting><NL>

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

See Also

- ":FUNCtion<m>:OPERation" on page 456
- ":FUNCtion<m>:INTegrate:IOFFset" on page 451

:FUNCtion<m>:INTegrate:IOFFset

 (see page 1292)

Command Syntax

```
:FUNCtion<m>:INTegrate:IOFFset <input_offset>

<m> ::= 1 to (# math functions) in NR1 format

<input_offset> ::= DC offset correction in NR3 format.
```

The :FUNCtion<m>:INTegrate:IOFFset command lets you enter a DC offset correction factor for the integrate math waveform input signal. This DC offset correction lets you level a "ramp"ed waveform.

Query Syntax

```
:FUNCtion<m>:INTegrate:IOFFset?
```

The :FUNCtion<m>:INTegrate:IOFFset? query returns the current input offset value.

Return Format

```
<input_offset><NL>

<input_offset> ::= DC offset correction in NR3 format.
```

See Also

- "Introduction to :FUNCtion<m> Commands" on page 422
- ":FUNCtion<m>:OPERation" on page 456
- ":FUNCtion<m>:INTegrate:IICondition" on page 450

:FUNCtion<m>:LABel

(see page 1292)

Command Syntax

```
:FUNCtion<m>:LABel <string>

<m> ::= 1 to (# math functions) in NR1 format

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

NOTE Label strings are 32 characters or less, and may contain any commonly used ASCII characters. Labels with more than 32 characters are truncated to 32 characters.

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

Query Syntax

```
:FUNCtion<m>:LABel?
```

The :FUNCtion<m>:LABel? query returns the label associated with a particular math function.

Return Format

```
<string><NL>

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

See Also

- "Introduction to :FUNCtion<m> Commands" on page 422
- ":FUNCtion<m>:DISPlay" on page 431
- ":DISPlay:LABel" on page 350
- ":DISPlay:LABList" on page 351
- ":VIEW" on page 227
- ":BLANk" on page 217
- ":STATus?" on page 224

:FUNCtion<m>:LINear:GAIN

N (see page 1292)

Command Syntax

```
:FUNCtion<m>:LINear:GAIN <value>

<m> ::= 1 to (# math functions) in NR1 format

<value> ::= 'A' in Ax + B, value in NR3 format
```

The :FUNCtion<m>:LINear:GAIN command specifies the 'A' value in the Ax + B operation.

Query Syntax

```
:FUNCtion<m>:LINear:GAIN?
```

The :FUNCtion<m>:LINear:GAIN query returns the gain value.

Return Format

```
<value><NL>

<value> ::= 'A' in Ax + B, value in NR3 format
```

See Also

- ":FUNCtion<m>:OPERation" on page 456

:FUNCtion<m>:LINear:OFFSet

(see page 1292)

Command Syntax

```
:FUNCtion<m>:LINear:OFFSet <value>

<m> ::= 1 to (# math functions) in NR1 format

<value> ::= 'B' in Ax + B, value in NR3 format
```

The :FUNCtion<m>:LINear:OFFSet command specifies the 'B' value in the Ax + B operation.

Query Syntax

```
:FUNCtion<m>:LINear:OFFSet?
```

The :FUNCtion<m>:LINear:OFFSet query returns the offset value.

Return Format

```
<value><NL>

<value> ::= 'B' in Ax + B, value in NR3 format
```

See Also

- ":FUNCtion<m>:OPERation" on page 456

:FUNCtion<m>:OFFSet

 (see page 1292)

Command Syntax

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

<m> ::= 1 to (# math functions) in NR1 format

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

The :FUNCtion<m>: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<m>:OFFSet command is equivalent to the :FUNCtion<m>:REFerence command.**

Query Syntax

```
:FUNCtion<m>:OFFSet?
```

The :FUNCtion<m>: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<m> Commands" on page 422
- ":FUNCtion<m>:RANGe" on page 460
- ":FUNCtion<m>:REFerence" on page 461
- ":FUNCtion<m>:SCALe" on page 462

:FUNCtion<m>:OPERation

N (see page 1292)

Command Syntax

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

<m> ::= 1 to (# math functions) in NR1 format

<operation> ::= {ADD | SUBTract | MULTiply | DIVide | DIFF | INTegrate
   | FFT | FFTPhase | SQRT | MAGNify | ABSolute | SQUare | LN | LOG
   | EXP | TEN | LOWPass | HIGHpass | BANDpass | AVERage | SMOoth
    | ENVelope | LINear | MAXimum | MINimum | MAXHold | MINHold
    | TRENd}
```

The :FUNCtion<m>:OPERation command sets the desired waveform math operator, transform, filter or visualization:

- Operators:
  - ADD – Source1 + source2.
  - SUBTract – Source1 - source2.
  - MULTiply – Source1 * source2.
  - DIVide – Source1 / source2.

  Operators perform their function on two analog channel sources.
- Transforms:
  - DIFF – Differentiate
  - INTegrate – The :FUNCtion<m>:INTegrate:IOFFset command lets you specify a DC offset correction factor. The :FUNCtion<m>:INTegrate:IICondition command lets you sets the Initial Condition of the Integrate math function waveform.
  - FFT (magnitude) – Using the Fast Fourier Transform (FFT), this operation displays the magnitudes of the frequency content that makes up the source waveform. The FFT takes the digitized time record of the specified source and transforms it to the frequency domain.

    The SPAN, CENTer, VTYPe, and WINDow commands are used for FFT functions. When FFT is selected, the horizontal cursors change from time to frequency (Hz), and the vertical cursors change from volts to decibels or V RMS.
  - FFTPhase – Using the Fast Fourier Transform (FFT), this operation shows the phase relationships of the frequency content that makes up the source waveform. The FFT takes the digitized time record of the specified source and transforms it to the frequency domain.

    The SPAN, CENTer, VTYPe, and WINDow commands are used for FFT functions. When FFTPhase is selected, the horizontal cursors change from time to frequency (Hz), and the vertical cursors change from volts to degrees or radians.

- LINear — Ax + B — The LINear commands set the gain (A) and offset (B) values for this function.
- SQUare
- SQRT — Square root
- ABSolute — Absolute Value
- LOG — Common Logarithm
- LN — Natural Logarithm
- EXP — Exponential (e^X)
- TEN — Base 10 exponential (10^X)

Transforms operate on a single analog channel source or on lower math functions.

- Filters:
  - LOWPass — Low pass filter — The FREQuency:LOWPass command sets the -3 dB cutoff frequency.
  - HIGHpass — High pass filter — The FREQuency:HIGHpass command sets the -3 dB cutoff frequency.
  - BANDpass — Band pass filter — The FREQuency:BANDpass:CENTer and FREQuency:BANDpass:WIDTh commands set the center frequency and width of the filter.
  - AVERage — Averaged value — The AVERage:COUNt command specifies the number of averages.

    Unlike acquisition averaging, the math averaging operator can be used to average the data on a single analog input channel or math function.

    If acquisition averaging is also used, the analog input channel data is averaged and the math function averages it again. You can use both types of averaging to get a certain number of averages on all waveforms and an increased number of averages on a particular waveform.

    Averages are calculated using a "decaying average" approximation, where:

    next_average = current_average + (new_data - current_average)/N

    Where N starts at 1 for the first acquisition and increments for each following acquisition until it reaches the selected number of averages, where it holds.
  - SMOoth — Smoothing — The resulting math waveform is the selected source with a normalized rectangular (boxcar) FIR filter applied.

    The boxcar filter is a moving average of adjacent waveform points, where the number of adjacent points is specified by the SMOoth:POINts command. You can choose an odd number of points, from 3 to 999.

The smoothing operator limits the bandwidth of the source waveform. The smoothing operator can be used, for example, to smooth measurement trend waveforms.

- ENVelope — Envelope — The resulting math waveform is the amplitude envelope for an amplitude modulated (AM) input signal.

  This function uses a Hilbert transform to get the real (in-phase, I) and imaginary (quadrature, Q) parts of the input signal and then performs a square root of the sum of the real and imaginary parts to get the demodulated amplitude envelope waveform.

Filters operate on a single analog channel source or on a lower math function.

- Visualizations:
  - MAGNify — Operates on a single analog channel source or on a lower math function.
  - MAXimum — This operator is like the MAXHold operator without the hold. The maximum vertical values found at each horizontal bucket are used to build a waveform.
  - MINimum — This operator is like the MINHold operator without the hold. The minimum vertical values found at each horizontal bucket are used to build a waveform.
  - MAXHold — Operates on a single analog channel source or on a lower math function. The Max Hold (or Max Envelope) operator records the maximum vertical values found at each horizontal bucket across multiple analysis cycles and uses those values to build a waveform.
  - MINHold — Operates on a single analog channel source or on a lower math function. The Min Hold (or Min Envelope) operator records the minimum vertical values found at each horizontal bucket across multiple analysis cycles and uses those values to build a waveform.
  - TRENd — Measurement trend — Operates on a single analog channel source. The TRENd:NMEasurement command selects the measurement whose trend you want to measure.

NOTE **If a math function is on (see :FUNCtion<m>:DISPlay), changing the operator will cause an automatic vertical scaling of the new math function waveform. If you wait for the operation to complete (with an *OPC? query for example), then query the math function's scale, you can see the vertical scaling that was automatically determined.**

Query Syntax

```
:FUNCtion<m>:OPERation?
```

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

Return Format

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

```
<operation> ::= {ADD | SUBT | MULT | DIV | INT | DIFF | FFT | FFTP
    | SQRT | MAGN | ABS | SQU | LN | LOG | EXP | TEN | LOWP | HIGH
     | BAND | AVER | SMO | ENV | LIN | MAX | MIN | MAXH | MINH
     | TREN}
```

See Also

- "Introduction to :FUNCtion<m> Commands" on page 422
- ":FUNCtion<m>:DISPlay" on page 431
- "*OPC (Operation Complete)" on page 191
- ":FUNCtion<m>:SOURce1" on page 464
- ":FUNCtion<m>:SOURce2" on page 466
- ":FUNCtion<m>:INTegrate:IICondition" on page 450
- ":FUNCtion<m>:INTegrate:IOFFset" on page 451
- ":FUNCtion<m>[:FFT]:SPAN" on page 442
- ":FUNCtion<m>[:FFT]:CENTer" on page 433
- ":FUNCtion<m>[:FFT]:PHASe:REFerence" on page 439
- ":FUNCtion<m>[:FFT]:VTYPe" on page 444
- ":FUNCtion<m>[:FFT]:WINDow" on page 445
- ":FUNCtion<m>:LINear:GAIN" on page 453
- ":FUNCtion<m>:LINear:OFFSet" on page 454
- ":FUNCtion<m>:FREQuency:BANDpass:CENTer" on page 446
- ":FUNCtion<m>:FREQuency:BANDpass:WIDTh" on page 447
- ":FUNCtion<m>:FREQuency:LOWPass" on page 449
- ":FUNCtion<m>:FREQuency:HIGHpass" on page 448
- ":FUNCtion<m>:AVERage:COUNt" on page 424
- ":FUNCtion<m>:SMOoth:POINts" on page 463
- ":FUNCtion<m>:TRENd:NMEasurement" on page 467
- ":FUNCtion<m>:BUS:YINCrement" on page 427
- ":FUNCtion<m>:BUS:YORigin" on page 428
- ":FUNCtion<m>:BUS:YUNits" on page 429
- ":FUNCtion<m>:BUS:CLOCk" on page 425
- ":FUNCtion<m>:BUS:SLOPe" on page 426

:FUNCtion<m>:RANGe

(see page 1292)

Command Syntax

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

<m> ::= 1 to (# math functions) in NR1 format

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

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

Query Syntax

```
:FUNCtion<m>:RANGe?
```

The :FUNCtion<m>: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.
```

See Also

- "Introduction to :FUNCtion<m> Commands" on page 422
- ":FUNCtion<m>:SCALe" on page 462

:FUNCtion<m>:REFerence

 (see page 1292)

Command Syntax

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

<m> ::= 1 to (# math functions) in NR1 format

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

The :FUNCtion<m>:REFerence 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 reference level to a value outside of the legal range, the level is automatically set to the nearest legal value.

NOTE **The FUNCtion:REFerence command is equivalent to the :FUNCtion<m>:OFFSet command.**

Query Syntax

```
:FUNCtion<m>:REFerence?
```

The :FUNCtion<m>:REFerence? query outputs the current reference level value for the selected function.

Return Format

```
<level><NL>

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

See Also

- "Introduction to :FUNCtion<m> Commands" on page 422
- ":FUNCtion<m>:OFFSet" on page 455
- ":FUNCtion<m>:RANGe" on page 460
- ":FUNCtion<m>:SCALe" on page 462

:FUNCtion<m>:SCALe

 (see page 1292)

Command Syntax

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

<m> ::= 1 to (# math functions) in NR1 format

<scale value> ::= vertical units/div value in NR3 format

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

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

NOTE Automatic vertical scaling of the math function waveform occurs when the function display is turned from off to on (see :FUNCtion<m>:DISPlay) or, if the function is already on, when the operation is changed (see :FUNCtion<m>:OPERation). If you want to change the math function's vertical scaling, you should do it after the math function display is turned on or after the operation is changed.

Query Syntax

```
:FUNCtion<m>:SCALe?
```

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

Return Format

```
<scale value><NL>

<scale value> ::= vertical units/div value in NR3 format
```

See Also

- "Introduction to :FUNCtion<m> Commands" on page 422
- ":FUNCtion<m>:DISPlay" on page 431
- ":FUNCtion<m>:OPERation" on page 456
- ":FUNCtion<m>:RANGe" on page 460

:FUNCtion<m>:SMOoth:POINts

N (see page 1292)

Command Syntax

```
:FUNCtion<m>:SMOoth:POINts <points>

<points> ::= odd integer in NR1 format
```

When the :FUNCtion<m>:OPERation is SMOoth, the :FUNCtion<m>:SMOoth:POINts command sets the number of smoothing points to use.

You can choose an odd number of points, from three up to half of the analysis record.

Query Syntax

```
:FUNCtion<m>:SMOoth:POINts?
```

The :FUNCtion<m>:SMOoth:POINts? query returns the number of smoothing points specified.

Return Format

```
<points><NL>
```

See Also

- ":FUNCtion<m>:OPERation" on page 456
- ":SYSTem:PRECision:LENGth" on page 1043

:FUNCtion<m>:SOURce1

N (see page 1292)

Command Syntax

```
:FUNCtion<m>:SOURce1 <value>

<m> ::= 1 to (# math functions) in NR1 format

<value> ::= {CHANnel<n> | FUNCtion<c> | MATH<c> | WMEMory<r> | BUS<b>}

<n> ::= 1 to (# analog channels) in NR1 format

<c> ::= {1 | 2 | 3}, must be lower than <m>

<r> ::= 1 to (# ref waveforms) in NR1 format

<b> ::= {1 | 2}
```

The :FUNCtion<m>:SOURce1 command is used for any :FUNCtion<m>:OPERation selection. This command selects the first source for the operator math functions or the single source for the transform functions, filter functions, or visualization functions.

The FUNCtion<c> or MATH<c> parameters are available for the transform functions, filter functions, and the magnify visualization function (see "Introduction to :FUNCtion<m> Commands" on page 422) when <c> is lower than <m>.

In other words, higher math functions can operate on lower math functions when using operators other than the simple arithmetic operations (+, - , *, /). For example, if :FUNCtion1:OPERation is a SUBTract of CHANnel1 and CHANnel2, the :FUNCtion2:OPERation could be set up as a FFT operation on the FUNCtion1 source. These are called cascaded math functions.

To cascade math functions, select the lower math function using the :FUNCtion<m>:SOURce1 command.

When cascading math functions, to get the most accurate results, be sure to vertically scale lower math functions so that their waveforms take up the full screen without being clipped.

The BUS<m> parameter is available for the bus charting visualization functions.

NOTE **Another shorthand notation for SOURce1 in this command/query (besides SOUR1) is SOUR.**

When :FUNCtion<m>:OPERation is TRENd, the :FUNCtion<m>:SOURce1 command reports error -221,"Settings conflict" because the TRENd function operates on a measurement and not a source waveform.

Query Syntax

```
:FUNCtion<m>:SOURce1?
```

The :FUNCtion<m>:SOURce1? query returns the current source1 for function operations.

When :FUNCtion<m>:OPERation is TRENd, the :FUNCtion<m>:SOURce1? query returns the source of the measurement.

Return Format

```
<value><NL>

<value> ::= {CHAN<n> | FUNC<c> | MATH<c> | WMEM<r> | BUS<b>}
```

See Also

- "Introduction to :FUNCtion<m> Commands" on page 422
- ":FUNCtion<m>:OPERation" on page 456

:FUNCtion<m>:SOURce2

N (see page 1292)

Command Syntax

```
:FUNCtion<m>:SOURce2 <value>

<m> ::= 1 to (# math functions) in NR1 format

<value> ::= {CHANnel<n> | WMEMory<r> | NONE}

<n> ::= 1 to (# analog channels) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :FUNCtion<m>:SOURce2 command specifies the second source for math operator functions that have two sources. (The :FUNCtion<m>:SOURce1 command specifies the first source.)

The :FUNCtion<m>:SOURce2 setting is not used for the transform functions, filter functions, or visualization functions (except when the measurement trend visualization's measurement requires two sources).

When :FUNCtion<m>:OPERation is TRENd, the :FUNCtion<m>:SOURce2 command reports error -221,"Settings conflict" because the TRENd function operates on a measurement and not a source waveform.

Query Syntax

```
:FUNCtion<m>:SOURce2?
```

The :FUNCtion<m>:SOURce2? query returns the currently specified second source for math operations.

When :FUNCtion<m>:OPERation is TRENd, the :FUNCtion<m>:SOURce2? query returns the source of the measurement.

Return Format

```
<value><NL>

<value> ::= {CHAN<n> | WMEM<r> | NONE}
```

See Also

- "Introduction to :FUNCtion<m> Commands" on page 422
- ":FUNCtion<m>:OPERation" on page 456
- ":FUNCtion<m>:SOURce1" on page 464

:FUNCtion<m>:TRENd:NMEasurement

N (see page 1292)

Command Syntax

```
:FUNCtion<m>:TRENd:NMEasurement MEAS<n>

<n> ::= # of installed measurement, from 1 to 8

<m> ::= 1 to (# math functions) in NR1 format
```

The :FUNCtion<m>:TRENd:NMEasurement command selects the measurement whose trend is shown in the math waveform.

There are 8 locations (or slots) that installed measurements can occupy. The MEAS<n> parameter specifies the location of the measurement whose trend you want to analyze.

You can view the trend math function waveform for these installed measurements:

- :MEASure:VAVerage, <interval> ::= CYCLe
- :MEASure:VRMS, <type> ::= AC (AC RMS)
- :MEASure:VRATio, <interval> ::= CYCLe
- :MEASure:PERiod
- :MEASure:FREQuency
- :MEASure:PWIDth
- :MEASure:NWIDth
- :MEASure:DUTYcycle
- :MEASure:NDUTy
- :MEASure:RISetime
- :MEASure:FALLtime

Query Syntax

```
:FUNCtion<m>:TRENd:NMEasurement?
```

The :FUNCtion<m>:TRENd:NMEasurement? query returns the selected measurement.

If no measurements are installed, the :FUNCtion<m>:TRENd:NMEasurement? query will return NONE.

Return Format

```
MEAS<n><NL>

<n> ::= # of installed measurement, from 1 to 8
```

See Also

- ":FUNCtion<m>:OPERation" on page 456

20 :HCOPy Commands

Commands for getting screen image data.

Table 85 :HCOPy Commands Summary

Command	Query	Options and Query Returns
n/a	:HCOPy:SDUMp:DATA? (see page 470)	<display_data> ::= binary block data in IEEE-488.2 # format.
:HCOPy:SDUMp[:DATA]:FORMat <format> (see page 471)	:HCOPy:SDUMp[:DATA]:FORMat? (see page 471)	<format> ::= {BMP | PNG}

:HCOPy:SDUMp:DATA?

(see page 1292)

Query Syntax

```
:HCOPy:SDUMp:DATA? [<format>]

<format> ::= {PNG | BMP}
```

The :HCOPy:SDUMp:DATA? query reads and returns screen image data. You can choose 24-bit BMP or 24-bit PNG formats.

In addition to the <format> option of this query, the screen image data format can also be set by the :HCOPy:SDUMp:DATA:FORMat command or the image **Format** selected in the graphical user interface's Save dialog box (**File > Save...**).

If no <format> option is specified with this query, the screen image data is returned in the currently selected format. After a *RST (factory default) command, the PNG format is selected by default.

If the <format> option is specified with this query, the format setting will affect the graphical user interface's **Format** selection in the Save dialog box.

Screen image data is returned in the IEEE-488.2 # binary block data format.

Return Format

```
<display_data><NL>

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

See Also

- ":HCOPy:SDUMp[:DATA]:FORMat" on page 471
- ":DISPlay:DATA?" on page 339

:HCOPy:SDUMp[:DATA]:FORMat

N (see page 1292)

Command Syntax

```
:HCOPy:SDUMp[:DATA]:FORMat <format>

<format> ::= {PNG | BMP}
```

The :HCOPy:SDUMp[:DATA]:FORMat command specifies the format for screen image data: 24-bit PNG or 24-bit BMP.

The :HCOPy:SDUMp[:DATA]:FORMat setting will persist when cycling power but will be reset to PNG after a *RST (factory default) command.

The :HCOPy:SDUMp[:DATA]:FORMat setting will affect the graphical user interface's **Format** selection in the Save dialog box (**File > Save...**).

Query Syntax

```
:HCOPy:SDUMp[:DATA]:FORMat?
```

The :HCOPy:SDUMp[:DATA]:FORMat? query returns the specified screen image data format.

The screen image data format can be set by the :HCOPy:SDUMp[:DATA]:FORMat command or the image **Format** selected in the graphical user interface's Save dialog box.

Return Format

```
<format><NL>

<format> ::= {PNG | BMP}
```

See Also

- ":HCOPy:SDUMp:DATA?" on page 470
- ":DISPlay:DATA?" on page 339
- "*RST (Reset)" on page 194

21 :HISTogram Commands

These commands control the histogram analysis feature.

The histogram analysis feature provides a statistical view of a waveform or a measurement's results.

Waveform histograms show the number of times a waveform crosses into (or hits) a row or column of a user-defined window. For vertical histograms, the window is divided into rows. For horizontal histograms, the window is divided into columns.

Measurement histograms show the distribution of a measurement's results similar to ordinary measurement statistics.

As waveforms are acquired and displayed, or as measurements are made, a counter database accumulates the total number of hits (or measurements), and the histogram data is displayed as a bar graph on the graticule.

When the oscilloscope is stopped, that counter database remains intact until the acquired data display is modified, usually horizontally or vertically. At that time, the database is reset and the last displayed acquisition is used to start a new database.

Table 86 :HISTogram Commands Summary

Command	Query	Options and Query Returns
:HISTogram:AXIS <axis> (see page 476)	:HISTogram:AXIS? (see page 476)	<axis> ::= {VERTical | HORizontal}
:HISTogram:DISPlay {{0 | OFF} | {1 | ON}} (see page 477)	:HISTogram:DISPlay? (see page 477)	{0 | 1}
:HISTogram:MEASurement MEAS<n> (see page 478)	:HISTogram:MEASurement? (see page 478)	<n> ::= # of installed measurement, from 1 to 10
:HISTogram:MODE <mode> (see page 479)	:HISTogram:MODE? (see page 479)	<mode> ::= {OFF | WAVeform | MEASurement}
:HISTogram:RESet (see page 480)	n/a	n/a

Table 86 :HISTogram Commands Summary (continued)

Command	Query	Options and Query Returns
:HISTogram:SIZE <size> (see page 481)	:HISTogram:SIZE? (see page 481)	<size> ::= 1.0 to 5.0 for vertical histograms, 1.0 to 4.0 for horizontal histograms, in NR3 format
:HISTogram:TYPE <type> (see page 482)	:HISTogram:TYPE? (see page 482)	<type> ::= {VERTical | HORizontal | MEASurement}
:HISTogram:WINDow:BLIMit <limit> (see page 483)	:HISTogram:WINDow:BLIMit? (see page 483)	<limit> ::= bottom value in NR3 format
:HISTogram:WINDow:LLIMit <limit> (see page 484)	:HISTogram:WINDow:LLIMit? (see page 484)	<limit> ::= left value in NR3 format
:HISTogram:WINDow:RLIMit <limit> (see page 485)	:HISTogram:WINDow:RLIMit? (see page 485)	<limit> ::= right value in NR3 format
:HISTogram:WINDow:SOURce <source> (see page 486)	:HISTogram:WINDow:SOURce? (see page 486)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format
:HISTogram:WINDow:TLIMit <limit> (see page 487)	:HISTogram:WINDow:TLIMit? (see page 487)	<limit> ::= top value in NR3 format

Histogram results statistics can be queried with these commands:

- ":MEASure:HISTogram:HITS?" on page 565 – The sum of all hits in all bins (buckets) in the histogram.
- ":MEASure:HISTogram:PEAK?" on page 574 – Maximum number of hits in any single bin.
- ":MEASure:HISTogram:MAXimum?" on page 569 – The value corresponding to the maximum bin that has any hits.
- ":MEASure:HISTogram:MINimum?" on page 572 – The value corresponding to the minimum bin that has any hits.
- ":MEASure:HISTogram:PPEak?" on page 575 – Delta between the Max and Min values.

- ":MEASure:HISTogram:MEAN?" on page 570 – Mean value of the histogram.
- ":MEASure:HISTogram:MEDian?" on page 571 – Median value of the histogram.
- ":MEASure:HISTogram:MODE?" on page 573 – Mode value of the histogram.
- ":MEASure:HISTogram:BWIDth?" on page 564 – The width of each bin (bucket) in the histogram.
- ":MEASure:HISTogram:SDEViation?" on page 576 – Standard deviation of the histogram values.
- ":MEASure:HISTogram:M1S?" on page 566 – Percentage of histogram hits that lie within ±1 Std Dev of the mean.
- ":MEASure:HISTogram:M2S?" on page 567 – Percentage of histogram hits that lie within ±2 Std Dev of the mean.
- ":MEASure:HISTogram:M3S?" on page 568 – Percentage of histogram hits that lie within ±3 Std Dev of the mean.

Histogram Interaction With Other Oscilloscope Features

When you enable a histogram, horizontal zoom is automatically disabled.

When using segmented memory acquisitions, histograms perform similarly to math waveforms and measurements. The histogram is reset at the start of a segmented acquisition and all segments are histogrammed at the end of the acquisition.

Because it is not possible to histogram every waveform, be aware that persistence can show a peak or glitch that does not show up in the histogram data.

Histograms do not wait for the specified number of averaged acquisitions to be acquired before starting analysis, so intermediate results could be erratic.

:HISTogram:AXIS

(see page 1292)

Command Syntax

```
:HISTogram:AXIS <axis>

<axis> ::= {VERTical | HORizontal}
```

When the :HISTogram:MODE is WAVeform, the :HISTogram:AXIS command specifies the orientation of the histogram:

- HORizontal — the limits window is divided into columns, and the number of hits in each column is displayed in a histogram bar graph at the bottom of the graticule.
- VERTical — the limits window is divided into rows, and the number of hits in each row is displayed in a histogram bar graph at the left of the graticule.

(When the :HISTogram:MODE is MEASurement, histograms are always displayed horizontally at the bottom of the graticule.)

Query Syntax

```
:HISTogram:AXIS?
```

The :HISTogram:AXIS? query returns the currently specified type of waveform histogram.

Return Format

```
<axis><NL>

<axis> ::= {VERT | HOR}
```

See Also

- ":HISTogram:DISPlay" on page 477
- ":HISTogram:TYPE" on page 482
- ":HISTogram:MODE" on page 479
- ":HISTogram:RESet" on page 480
- ":HISTogram:SIZE" on page 481
- ":HISTogram:WINDow:BLIMit" on page 483
- ":HISTogram:WINDow:LLIMit" on page 484
- ":HISTogram:WINDow:RLIMit" on page 485
- ":HISTogram:WINDow:SOURce" on page 486
- ":HISTogram:WINDow:TLIMit" on page 487

:HISTogram:DISPlay

(see page 1292)

Command Syntax `:HISTogram:DISPlay {{0 | OFF} | {1 | ON}}`

The :HISTogram:DISPlay command turns the histogram display on or off.

Query Syntax `:HISTogram:DISPlay?`

The :HISTogram:DISPlay? query returns the current display setting for the histogram.

Return Format `<off_on><NL>`

`{0 | 1}`

See Also

- ":HISTogram:TYPE" on page 482
- ":HISTogram:RESet" on page 480
- ":HISTogram:AXIS" on page 476
- ":HISTogram:MEASurement" on page 478
- ":HISTogram:MODE" on page 479
- ":HISTogram:SIZE" on page 481
- ":HISTogram:WINDow:BLIMit" on page 483
- ":HISTogram:WINDow:LLIMit" on page 484
- ":HISTogram:WINDow:RLIMit" on page 485
- ":HISTogram:WINDow:SOURce" on page 486
- ":HISTogram:WINDow:TLIMit" on page 487

:HISTogram:MEASurement

 (see page 1292)

Command Syntax `:HISTogram:MEASurement MEAS<n>`

`<n> ::= # of installed measurement, from 1 to 10`

When the :HISTogram:MODE is MEASurement, the :HISTogram:MEASurement command selects the measurement to analyze.

There are 10 locations (or slots) that installed measurements can occupy. The MEAS<n> parameter specifies the location of the measurement you want to analyze.

Query Syntax `:HISTogram:MEASurement?`

The :HISTogram:MEASurement? query returns the location of the measurement currently selected for histogram analysis.

Return Format `MEAS<n><NL>`

`<n> ::= # of installed measurement, from 1 to 10`

See Also

- ":HISTogram:DISPlay" on page 477
- ":HISTogram:TYPE" on page 482
- ":HISTogram:MODE" on page 479
- ":HISTogram:RESet" on page 480
- ":HISTogram:SIZE" on page 481

:HISTogram:MODE

(see page 1292)

Command Syntax

```
:HISTogram:MODE <mode>

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

The :HISTogram:MODE command selects the type of histogram or disables the histogram analysis feature:

- OFF – disables the histogram analysis feature.
- WAVeform – shows the number of times a waveform crosses into (or hits) a row or column of a user-defined window. For vertical histograms, the window is divided into rows. For horizontal histograms, the window is divided into columns. Use the :HISTogram:AXIS command to specify a vertical or horizontal histogram.
- MEASurement – shows the distribution of a measurement's results just like ordinary measurement statistics.

Query Syntax

```
:HISTogram:MODE?
```

The :HISTogram:MODE? query returns the currently set histogram mode.

Return Format

```
<mode><NL>

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

See Also

- ":HISTogram:DISPlay" on page 477
- ":HISTogram:TYPE" on page 482
- ":HISTogram:AXIS" on page 476
- ":HISTogram:MEASurement" on page 478
- ":HISTogram:RESet" on page 480
- ":HISTogram:SIZE" on page 481
- ":HISTogram:WINDow:BLIMit" on page 483
- ":HISTogram:WINDow:LLIMit" on page 484
- ":HISTogram:WINDow:RLIMit" on page 485
- ":HISTogram:WINDow:SOURce" on page 486
- ":HISTogram:WINDow:TLIMit" on page 487

:HISTogram:RESet

(see page 1292)

Command Syntax `:HISTogram:RESet`

The :HISTogram:RESet command zeros the histogram counters.

See Also

- ":HISTogram:DISPlay" on page 477
- ":HISTogram:TYPE" on page 482
- ":HISTogram:AXIS" on page 476
- ":HISTogram:MEASurement" on page 478
- ":HISTogram:MODE" on page 479
- ":HISTogram:SIZE" on page 481
- ":HISTogram:WINDow:BLIMit" on page 483
- ":HISTogram:WINDow:LLIMit" on page 484
- ":HISTogram:WINDow:RLIMit" on page 485
- ":HISTogram:WINDow:SOURce" on page 486
- ":HISTogram:WINDow:TLIMit" on page 487

:HISTogram:SIZE

(see page 1292)

Command Syntax

```
:HISTogram:SIZE <size>

<size> ::= 1.0 to 5.0 for vertical histograms, 1.0 to 4.0 for horizontal
   histograms, in NR3 format
```

The :HISTogram:SIZE command specifies the number of divisions the histogram bar graph should use:

- For horizontal histograms, you can choose whole and half divisions from 1 to 4.
- For vertical histograms, you can choose whole and half divisions from 1 to 5.

Query Syntax

```
:HISTogram:SIZE?
```

The :HISTogram:SIZE? query returns the number of divisions currently used for the histogram bar graph.

Return Format

```
<size><NL>

<size> ::= 1.0 to 5.0 for vertical histograms, 1.0 to 4.0 for horizontal
   histograms, in NR3 format
```

See Also

- ":HISTogram:DISPlay" on page 477
- ":HISTogram:TYPE" on page 482
- ":HISTogram:AXIS" on page 476
- ":HISTogram:MEASurement" on page 478
- ":HISTogram:MODE" on page 479
- ":HISTogram:RESet" on page 480
- ":HISTogram:WINDow:BLIMit" on page 483
- ":HISTogram:WINDow:LLIMit" on page 484
- ":HISTogram:WINDow:RLIMit" on page 485
- ":HISTogram:WINDow:SOURce" on page 486
- ":HISTogram:WINDow:TLIMit" on page 487

:HISTogram:TYPE

(see page 1292)

Command Syntax

```
:HISTogram:TYPE <type>

<type> ::= {VERTical | HORizontal | MEASurement}
```

The :HISTogram:TYPE command selects either a waveform histogram (horizontal or vertical) or a measurement histogram:

- HORizontal – the limits window is divided into columns, and the number of hits in each column is displayed in a histogram bar graph at the bottom of the graticule.
- VERTical – the limits window is divided into rows, and the number of hits in each row is displayed in a histogram bar graph at the left of the graticule.
- MEASurement – shows the distribution of a measurement's results similar to ordinary measurement statistics.

Query Syntax

```
:HISTogram:TYPE?
```

The :HISTogram:TYPE? query returns the currently selected histogram type.

Return Format

```
<type><NL>

<type> ::= {VERT | HOR | MEAS}
```

See Also

- ":HISTogram:DISPlay" on page 477
- ":HISTogram:RESet" on page 480
- ":HISTogram:AXIS" on page 476
- ":HISTogram:MEASurement" on page 478
- ":HISTogram:MODE" on page 479
- ":HISTogram:SIZE" on page 481
- ":HISTogram:WINDow:BLIMit" on page 483
- ":HISTogram:WINDow:LLIMit" on page 484
- ":HISTogram:WINDow:RLIMit" on page 485
- ":HISTogram:WINDow:SOURce" on page 486
- ":HISTogram:WINDow:TLIMit" on page 487

:HISTogram:WINDow:BLIMit

(see page 1292)

Command Syntax

```
:HISTogram:WINDow:BLIMit <limit>

<limit> ::= bottom value in NR3 format
```

When the :HISTogram:MODE is WAVeform, the :HISTogram:WINDow:BLIMit command specifies the histogram limits window's bottom boundary.

The units are the same as the input waveform's vertical units. The input waveform is specified using the :HISTogram:WINDow:SOURce command.

Query Syntax

```
:HISTogram:WINDow:BLIMit?
```

The :HISTogram:WINDow:BLIMit? query returns the histogram limits window's bottom boundary.

Return Format

```
<limit><NL>

<limit> ::= bottom value in NR3 format
```

See Also

- ":HISTogram:DISPlay" on page 477
- ":HISTogram:TYPE" on page 482
- ":HISTogram:AXIS" on page 476
- ":HISTogram:MODE" on page 479
- ":HISTogram:RESet" on page 480
- ":HISTogram:SIZE" on page 481
- ":HISTogram:WINDow:LLIMit" on page 484
- ":HISTogram:WINDow:RLIMit" on page 485
- ":HISTogram:WINDow:SOURce" on page 486
- ":HISTogram:WINDow:TLIMit" on page 487

:HISTogram:WINDow:LLIMit

 (see page 1292)

Command Syntax

```
:HISTogram:WINDow:LLIMit <limit>

<limit> ::= left value in NR3 format
```

When the :HISTogram:MODE is WAVeform, the :HISTogram:WINDow:LLIMit command specifies the histogram limits window's left boundary.

The units are the same as the input waveform's horizontal units. The input waveform is specified using the :HISTogram:WINDow:SOURce command.

Query Syntax

```
:HISTogram:WINDow:LLIMit?
```

The :HISTogram:WINDow:LLIMit? query returns the histogram limits window's left boundary.

Return Format

```
<limit><NL>

<limit> ::= left value in NR3 format
```

See Also

- ":HISTogram:DISPlay" on page 477
- ":HISTogram:TYPE" on page 482
- ":HISTogram:AXIS" on page 476
- ":HISTogram:MODE" on page 479
- ":HISTogram:RESet" on page 480
- ":HISTogram:SIZE" on page 481
- ":HISTogram:WINDow:BLIMit" on page 483
- ":HISTogram:WINDow:RLIMit" on page 485
- ":HISTogram:WINDow:SOURce" on page 486
- ":HISTogram:WINDow:TLIMit" on page 487

:HISTogram:WINDow:RLIMit

(see page 1292)

Command Syntax

```
:HISTogram:WINDow:RLIMit <limit>

<limit> ::= right value in NR3 format
```

When the :HISTogram:MODE is WAVeform, the :HISTogram:WINDow:RLIMit command specifies the histogram limits window's right boundary.

The units are the same as the input waveform's horizontal units. The input waveform is specified using the :HISTogram:WINDow:SOURce command.

Query Syntax

```
:HISTogram:WINDow:RLIMit?
```

The :HISTogram:WINDow:RLIMit? query returns the histogram limits window's right boundary.

Return Format

```
<limit><NL>

<limit> ::= right value in NR3 format
```

See Also

- ":HISTogram:DISPlay" on page 477
- ":HISTogram:TYPE" on page 482
- ":HISTogram:AXIS" on page 476
- ":HISTogram:MODE" on page 479
- ":HISTogram:RESet" on page 480
- ":HISTogram:SIZE" on page 481
- ":HISTogram:WINDow:BLIMit" on page 483
- ":HISTogram:WINDow:LLIMit" on page 484
- ":HISTogram:WINDow:SOURce" on page 486
- ":HISTogram:WINDow:TLIMit" on page 487

:HISTogram:WINDow:SOURce

(see page 1292)

Command Syntax

```
:HISTogram:WINDow:SOURce <source>

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

<n> ::= 1 to (# analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

When the :HISTogram:MODE is WAVeform, the :HISTogram:WINDow:SOURce command specifies the waveform that is used as the histogram source.

Query Syntax

```
:HISTogram:WINDow:SOURce?
```

The :HISTogram:WINDow:SOURce? query returns the currently specified waveform histogram source.

Return Format

```
<source><NL>
```

See Also

- ":HISTogram:DISPlay" on page 477
- ":HISTogram:TYPE" on page 482
- ":HISTogram:AXIS" on page 476
- ":HISTogram:MODE" on page 479
- ":HISTogram:RESet" on page 480
- ":HISTogram:SIZE" on page 481
- ":HISTogram:WINDow:BLIMit" on page 483
- ":HISTogram:WINDow:LLIMit" on page 484
- ":HISTogram:WINDow:RLIMit" on page 485
- ":HISTogram:WINDow:TLIMit" on page 487

:HISTogram:WINDow:TLIMit

(see page 1292)

Command Syntax

```
:HISTogram:WINDow:TLIMit <limit>

<limit> ::= top value in NR3 format
```

When the :HISTogram:MODE is WAVeform, the :HISTogram:WINDow:TLIMit command specifies the histogram limits window's top boundary.

The units are the same as the input waveform's vertical units. The input waveform is specified using the :HISTogram:WINDow:SOURce command.

Query Syntax

```
:HISTogram:WINDow:TLIMit?
```

The :HISTogram:WINDow:TLIMit? query returns the histogram limits window's top boundary.

Return Format

```
<limit><NL>

<limit> ::= top value in NR3 format
```

See Also

- ":HISTogram:DISPlay" on page 477
- ":HISTogram:TYPE" on page 482
- ":HISTogram:AXIS" on page 476
- ":HISTogram:MODE" on page 479
- ":HISTogram:RESet" on page 480
- ":HISTogram:SIZE" on page 481
- ":HISTogram:WINDow:BLIMit" on page 483
- ":HISTogram:WINDow:LLIMit" on page 484
- ":HISTogram:WINDow:RLIMit" on page 485
- ":HISTogram:WINDow:SOURce" on page 486

22 :LISTer Commands

Table 87 :LISTer Commands Summary

Command	Query	Options and Query Returns
n/a	:LISTer:DATA? (see page 490)	<binary_block> ::= comma-separated data with newlines at the end of each row
:LISTer:DISPlay {{OFF | 0} | {SBUS1 | ON | 1} | {SBUS2 | 2}} (see page 491)	:LISTer:DISPlay? (see page 491)	{OFF | SBUS1 | SBUS2}
:LISTer:REFerence <time_ref> (see page 492)	:LISTer:REFerence? (see page 492)	<time_ref> ::= {TRIGger | PREVious}

Introduction to :LISTer Commands

The LISTer subsystem is used to turn on/off the serial decode Lister display and return data from the Lister display.

:LISTer:DATA?

(see page 1292)

Query Syntax `:LISTer:DATA?`

The :LISTer:DATA? query returns the lister data.

Return Format `<binary block><NL>`

```
<binary_block> ::= comma-separated data with newlines at the
                   end of each row
```

See Also

- "Introduction to :LISTer Commands" on page 489
- ":LISTer:DISPlay" on page 491
- "Definite-Length Block Response Data" on page 177

:LISTer:DISPlay

 (see page 1292)

Command Syntax

```
:LISTer:DISPlay <value>

<value> ::= {{OFF | 0} | {SBUS1 | ON | 1} | {SBUS2 | 2}}
```

The :LISTer:DISPlay command configures which of the serial buses to display in the Lister, or whether the Lister is off. "ON" or "1" is the same as "SBUS1".

Serial bus decode must be on before it can be displayed in the Lister.

Query Syntax

```
:LISTer:DISPlay?
```

The :LISTer:DISPlay? query returns the Lister display setting.

Return Format

```
<value><NL>

<value> ::= {OFF | SBUS1 | SBUS2}
```

See Also

- "Introduction to :LISTer Commands" on page 489
- ":SBUS<n>:DISPlay" on page 718
- ":LISTer:DATA?" on page 490

:LISTer:REFerence

(see page 1292)

Command Syntax

```
:LISTer:REFerence <time_ref>

<time_ref> ::= {TRIGger | PREVious}
```

The :LISTer:REFerence command selects whether the time value for a Lister row is relative to the trigger or the previous Lister row.

Query Syntax

```
:LISTer:REFerence?
```

The :LISTer:REFerence? query returns the Lister time reference setting.

Return Format

```
<time_ref><NL>

<time_ref> ::= {TRIGger | PREVious}
```

See Also

- "Introduction to :LISTer Commands" on page 489
- ":SBUS<n>:DISPlay" on page 718
- ":LISTer:DATA?" on page 490
- ":LISTer:DISPlay" on page 491

23 :LTESt Commands

Table 88 :LTESt Commands Summary

Command	Query	Options and Query Returns
:LTESt:COPY (see page 495)	n/a	n/a
:LTESt:COPY:ALL (see page 496)	n/a	n/a
:LTESt:COPY:MARGin <percent> (see page 497)	:LTESt:COPY:MARGin? (see page 497)	<percent> ::= copy margin percent in NR1 format
:LTESt:ENABle {{0 | OFF} | {1 | ON}} (see page 498)	:LTESt:ENABle? (see page 498)	<setting> ::= {0 | 1}
:LTESt:FAIL <condition> (see page 499)	:LTESt:FAIL? (see page 499)	<condition> ::= {INSide | OUTSide}
:LTESt:LLIMit <lower_value> (see page 500)	:LTESt:LLIMit? (see page 500)	<lower_value> ::= a real number in NR3 format
:LTESt:MEASurement {MEAS<n>} (see page 501)	:LTESt:MEASurement? (see page 501)	<n> ::= # of installed measurement, from 1 to 10
n/a	:LTESt:RESults? [{MEAS<n>}] (see page 502)	<n> ::= # of installed measurement, from 1 to 10
:LTESt:RUMode:SOFailure {{0 | OFF} | {1 | ON}} (see page 503)	:LTESt:RUMode:SOFailure? (see page 503)	<setting> ::= {0 | 1}

Table 88 :LTESt Commands Summary (continued)

Command	Query	Options and Query Returns
:LTESt:TEST {{0 | OFF} | {1 | ON}} (see page 504)	:LTESt:TEST? (see page 504)	<setting> ::= {0 | 1}
:LTESt:ULIMit <upper_value> (see page 505)	:LTESt:ULIMit? (see page 505)	<upper_value> ::= a real number in NR3 format

Introduction to :LTESt Commands

The LTESt subsystem is used for the Measurement Limit Test feature.

:LTESt:COPY

 (see page 1292)

Command Syntax `:LTESt:COPY`

For the measurement selected by :LTESt:MEASurement, the :LTESt:COPY command sets upper and lower limits to the last measurement result plus or minus a percent margin (see :LTESt:COPY:MARGin).

See Also

- ":LTESt:COPY:ALL" on page 496
- ":LTESt:COPY:MARGin" on page 497
- ":LTESt:ENABle" on page 498
- ":LTESt:FAIL" on page 499
- ":LTESt:LLIMit" on page 500
- ":LTESt:MEASurement" on page 501
- ":LTESt:RESults?" on page 502
- ":LTESt:RUMode:SOFailure" on page 503
- ":LTESt:TEST" on page 504
- ":LTESt:ULIMit" on page 505

:LTESt:COPY:ALL

 (see page 1292)

Command Syntax `:LTESt:COPY:ALL`

For all measurements whose limits are being tested, the :LTESt:COPY:ALL command sets upper and lower limits to the last measurement result plus or minus a percent margin (see :LTESt:COPY:MARGin).

In essence, this command performs the :LTESt:COPY operation on all measurements whose limits are being tested.

See Also

- ":LTESt:COPY" on page 495
- ":LTESt:COPY:MARGin" on page 497
- ":LTESt:ENABle" on page 498
- ":LTESt:FAIL" on page 499
- ":LTESt:LLIMit" on page 500
- ":LTESt:MEASurement" on page 501
- ":LTESt:RESults?" on page 502
- ":LTESt:RUMode:SOFailure" on page 503
- ":LTESt:TEST" on page 504
- ":LTESt:ULIMit" on page 505

:LTESt:COPY:MARGin

N (see page 1292)

Command Syntax

```
:LTESt:COPY:MARGin <percent>

<percent> ::= copy margin percent in NR1 format
```

For the measurement selected by :LTESt:MEASurement, the :LTESt:COPY:MARGin command specifies the percent margin used when setting upper and lower limits by copying from measurement results.

Query Syntax

```
:LTESt:COPY:MARGin?
```

The :LTESt:COPY:MARGin? query returns the percent margin setting for the measurement selected by :LTESt:MEASurement.

Return Format

```
<percent><NL>

<percent> ::= copy margin percent in NR1 format
```

See Also

- ":LTESt:COPY" on page 495
- ":LTESt:COPY:ALL" on page 496
- ":LTESt:ENABle" on page 498
- ":LTESt:FAIL" on page 499
- ":LTESt:LLIMit" on page 500
- ":LTESt:MEASurement" on page 501
- ":LTESt:RESults?" on page 502
- ":LTESt:RUMode:SOFailure" on page 503
- ":LTESt:TEST" on page 504
- ":LTESt:ULIMit" on page 505

:LTESt:ENABle

 (see page 1292)

Command Syntax `:LTESt:ENABle {{0 | OFF} | {1 | ON}}`

The :LTESt:ENABle command turns the measurement limit test feature on or off.

Query Syntax `:LTESt:ENABle?`

The :LTESt:ENABle? query returns whether the measurement limit test feature is on or off.

Return Format `<setting><NL>`

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

See Also

- ":LTESt:COPY" on page 495
- ":LTESt:COPY:ALL" on page 496
- ":LTESt:COPY:MARGin" on page 497
- ":LTESt:FAIL" on page 499
- ":LTESt:LLIMit" on page 500
- ":LTESt:MEASurement" on page 501
- ":LTESt:RESults?" on page 502
- ":LTESt:RUMode:SOFailure" on page 503
- ":LTESt:TEST" on page 504
- ":LTESt:ULIMit" on page 505

:LTESt:FAIL

(see page 1292)

Command Syntax

```
:LTESt:FAIL <condition>

<condition> ::= {INSide | OUTSide}
```

For the active measurement currently selected by the :LTESt:MEASurement command, the :LTESt:FAIL command sets the fail condition for the measurement.

When a measurement failure is detected by the limit test, the fail action conditions are executed, and there is the potential to generate an SRQ.

- INSide – causes the oscilloscope to fail a test when the measurement results are within the parameters set by the :LLTESt:LIMit and :LTESt:ULIMit commands.
- OUTSide – causes the oscilloscope to fail a test when the measurement results exceed the parameters set by :LTESt:LLIMit and :LTESt:ULIMit commands.

Query Syntax

```
:LTESt:FAIL?
```

The :LTESt:FAIL? query returns the currently set fail condition.

Return Format

```
<condition><NL>

<condition> ::= {INS | OUTS}
```

See Also

- ":LTESt:COPY" on page 495
- ":LTESt:COPY:ALL" on page 496
- ":LTESt:COPY:MARGin" on page 497
- ":LTESt:ENABle" on page 498
- ":LTESt:LLIMit" on page 500
- ":LTESt:MEASurement" on page 501
- ":LTESt:RESults?" on page 502
- ":LTESt:RUMode:SOFailure" on page 503
- ":LTESt:TEST" on page 504
- ":LTESt:ULIMit" on page 505

:LTESt:LLIMit

N (see page 1292)

Command Syntax

```
:LTESt:LLIMit <lower_value>

<lower_value> ::= a real number in NR3 format
```

For the active measurement currently selected by the :LTESt:MEASurement command, the :LTESt:LLIMit (Lower LIMit) command sets the lower test limit.

Query Syntax

```
:LTESt:LLIMit?
```

The :LTESt:LLIMit? query returns the current lower limit setting.

Return Format

```
<lower_value><NL>
```

See Also

- ":LTESt:COPY" on page 495
- ":LTESt:COPY:ALL" on page 496
- ":LTESt:COPY:MARGin" on page 497
- ":LTESt:ENABle" on page 498
- ":LTESt:FAIL" on page 499
- ":LTESt:MEASurement" on page 501
- ":LTESt:RESults?" on page 502
- ":LTESt:RUMode:SOFailure" on page 503
- ":LTESt:TEST" on page 504
- ":LTESt:ULIMit" on page 505

:LTESt:MEASurement

(see page 1292)

Command Syntax

```
:LTESt:MEASurement {MEAS<n>}

<n> ::= # of installed measurement, from 1 to 10
```

The :LTESt:MEASurement command selects the measurement source for the :LTESt:FAIL, :LTESt:LLIMit, :LTESt:ULIMit, and :LTESt:TEST commands. It selects one of the active measurements by its number, where MEAS1 is the first added measurement. When you add more than 10 measurements, measurement numbering starts again from MEAS1.

Query Syntax

```
:LTESt:MEASurement?
```

The :LTESt:MEASurement? query returns the currently selected measurement source.

Return Format

```
MEAS<n><NL>
```

See Also

- ":LTESt:COPY" on page 495
- ":LTESt:COPY:ALL" on page 496
- ":LTESt:COPY:MARGin" on page 497
- ":LTESt:ENABle" on page 498
- ":LTESt:FAIL" on page 499
- ":LTESt:LLIMit" on page 500
- ":LTESt:RESults?" on page 502
- ":LTESt:RUMode:SOFailure" on page 503
- ":LTESt:TEST" on page 504
- ":LTESt:ULIMit" on page 505

:LTESt:RESults?

(see page 1292)

Query Syntax

```
:LTESt:RESults? [{MEAS<n>}]

<n> ::= # of installed measurement, from 1 to 10
```

The :LTESt:RESults? query returns the measurement results for selected measurement.

When :LTESt:TEST is ON, the :LTESt:RESults? query returns the failed minimum value (Fail Min), the failed maximum value (Fail Max), and the total number of measurements made (# of Meas).

When :LTESt:TEST is OFF, the :LTESt:RESults? query returns nothing.

Return Format

```
<fail_min>,<fail_max>,<num_meas><NL>
```

- <fail_min> – A real number representing the total number of measurements that have failed the minimum limit.
- <fail_max> – A real number representing the total number of measurements that have failed the maximum limit.
- <num_meas> – A real number representing the total number of measurements that have been made.

See Also

- ":LTESt:COPY" on page 495
- ":LTESt:COPY:ALL" on page 496
- ":LTESt:COPY:MARGin" on page 497
- ":LTESt:ENABle" on page 498
- ":LTESt:FAIL" on page 499
- ":LTESt:LLIMit" on page 500
- ":LTESt:MEASurement" on page 501
- ":LTESt:RUMode:SOFailure" on page 503
- ":LTESt:TEST" on page 504
- ":LTESt:ULIMit" on page 505

:LTESt:RUMode:SOFailure

(see page 1292)

Command Syntax

```
:LTESt:RUMode:SOFailure {{0 | OFF} | {1 | ON}}
```

The :LTESt:RUMode:SOFailure command enables or disables the limit test "stop on failure" option.

When ON, the oscilloscope acquisition system stops once a limit failure is detected. If more than one measurement limit test is enabled, a failure of any of the measurements stops the oscilloscope from acquiring new waveforms.

Query Syntax

```
:LTESt:RUMode:SOFailure?
```

The :LTESt:RUMode:SOFailure? query returns returns the "stop on failure" setting.

Return Format

```
<setting><NL>

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

See Also

- ":LTESt:COPY" on page 495
- ":LTESt:COPY:ALL" on page 496
- ":LTESt:COPY:MARGin" on page 497
- ":LTESt:ENABle" on page 498
- ":LTESt:FAIL" on page 499
- ":LTESt:LLIMit" on page 500
- ":LTESt:MEASurement" on page 501
- ":LTESt:RESults?" on page 502
- ":LTESt:TEST" on page 504
- ":LTESt:ULIMit" on page 505

:LTESt:TEST

(see page 1292)

Command Syntax

```
:LTESt:TEST {{0 | OFF} | {1 | ON}}
```

For the active measurement currently selected by the :LTESt:MEASurement command, the :LTESt:TEST command enables or disables the limit test function on that measurement.

When any measurement has its limit test function enabled, the overall Limit Test feature is enabled.

The :LTESt:RESults? query returns nothing when :LTESt:TEST is OFF.

Query Syntax

```
:LTESt:TEST?
```

The :LTESt:TEST? query returns the state of the TEST control for the active measurement currently selected by the :LTESt:MEASurement command.

Return Format

```
<setting><NL>

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

See Also

- ":LTESt:COPY" on page 495
- ":LTESt:COPY:ALL" on page 496
- ":LTESt:COPY:MARGin" on page 497
- ":LTESt:ENABle" on page 498
- ":LTESt:FAIL" on page 499
- ":LTESt:LLIMit" on page 500
- ":LTESt:MEASurement" on page 501
- ":LTESt:RESults?" on page 502
- ":LTESt:RUMode:SOFailure" on page 503
- ":LTESt:ULIMit" on page 505

:LTESt:ULIMit

N (see page 1292)

Command Syntax

```
:LTESt:ULIMit <upper_value>

<upper_value> ::= a real number in NR3 format
```

For the active measurement currently selected by the :LTESt:MEASurement command, the :LTESt:ULIMit (Upper LIMit) command sets the upper test limit.

Query Syntax

```
:LTESt:ULIMit?
```

The :LTESt:ULIMit? query returns the current upper limit setting.

Return Format

```
<upper_value><NL>
```

See Also

- ":LTESt:COPY" on page 495
- ":LTESt:COPY:ALL" on page 496
- ":LTESt:COPY:MARGin" on page 497
- ":LTESt:ENABle" on page 498
- ":LTESt:FAIL" on page 499
- ":LTESt:LLIMit" on page 500
- ":LTESt:MEASurement" on page 501
- ":LTESt:RESults?" on page 502
- ":LTESt:RUMode:SOFailure" on page 503
- ":LTESt:TEST" on page 504

24 :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" on page 509.

Table 89 :MARKer Commands Summary

Command	Query	Options and Query Returns
n/a	:MARKer:DYDX? (see page 510)	<return_value> ::= ΔY/ΔX value in NR3 format
:MARKer:MODE <mode> (see page 511)	:MARKer:MODE? (see page 511)	<mode> ::= {OFF | MEASurement | MANual | WAVeform | BINary | HEX}
:MARKer:X1:DISPlay {{0 | OFF} | {1 | ON}} (see page 512)	:MARKer:X1:DISPlay? (see page 512)	<setting> ::= {0 | 1}
:MARKer:X1Position <position>[suffix] (see page 513)	:MARKer:X1Position? (see page 513)	<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> (see page 514)	:MARKer:X1Y1source? (see page 514)	<source> ::= {CHANnel<n> | FUNCtion<m> | FFT | MATH<m> | WMEMory<r> | HISTogram} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= <source>
:MARKer:X2:DISPlay {{0 | OFF} | {1 | ON}} (see page 515)	:MARKer:X2:DISPlay? (see page 515)	<setting> ::= {0 | 1}

Table 89 :MARKer Commands Summary (continued)

Command	Query	Options and Query Returns
:MARKer:X2Position <position>[suffix] (see page 516)	:MARKer:X2Position? (see page 516)	<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> (see page 517)	:MARKer:X2Y2source? (see page 517)	<source> ::= {CHANnel<n> | FUNCtion<m> | FFT | MATH<m> | WMEMory<r> | HISTogram} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= <source>
n/a	:MARKer:XDELta? (see page 518)	<return_value> ::= X cursors delta value in NR3 format
:MARKer:XUNits <mode> (see page 519)	:MARKer:XUNits? (see page 519)	<units> ::= {SEConds | HERTz | DEGRees | PERCent}
:MARKer:XUNits:USE (see page 520)	n/a	n/a
:MARKer:Y1:DISPlay {{0 | OFF} | {1 | ON}} (see page 521)	:MARKer:Y1:DISPlay? (see page 521)	<setting> ::= {0 | 1}
:MARKer:Y1Position <position>[suffix] (see page 522)	:MARKer:Y1Position? (see page 522)	<position> ::= Y1 cursor position value in NR3 format [suffix] ::= {V | mV | dB} <return_value> ::= Y1 cursor position value in NR3 format
:MARKer:Y2:DISPlay {{0 | OFF} | {1 | ON}} (see page 523)	:MARKer:Y2:DISPlay? (see page 523)	<setting> ::= {0 | 1}
:MARKer:Y2Position <position>[suffix] (see page 524)	:MARKer:Y2Position? (see page 524)	<position> ::= Y2 cursor position value in NR3 format [suffix] ::= {V | mV | dB} <return_value> ::= Y2 cursor position value in NR3 format

Table 89 :MARKer Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:MARKer:YDELta? (see page 525)	<return_value> ::= Y cursors delta value in NR3 format
:MARKer:YUNits <mode> (see page 526)	:MARKer:YUNits? (see page 526)	<units> ::= {BASE | PERCent}
:MARKer:YUNits:USE (see page 527)	n/a	n/a

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 CHAN1;X2Y2 CHAN1;MODE MAN
```

:MARKer:DYDX?

N (see page 1292)

Query Syntax `:MARKer:DYDX?`

The MARKer:DYDX? query returns the cursor ΔY/ΔX value.

X cursor units are set by the :MARKer:XUNits command.

NOTE If the front-panel cursors are off, the marker position values are not defined. Make sure to set :MARKer:MODE to MANual or WAVeform to put the cursors in the front-panel Normal mode.

Return Format `<value><NL>`

`<value> ::= ΔY/ΔX value in NR3 format.`

See Also
- "Introduction to :MARKer Commands" on page 509
- ":MARKer:MODE" on page 511
- ":MARKer:X1Position" on page 513
- ":MARKer:X2Position" on page 516
- ":MARKer:X1Y1source" on page 514
- ":MARKer:X2Y2source" on page 517
- ":MARKer:XUNits" on page 519

:MARKer:MODE

(see page 1292)

Command Syntax

```
:MARKer:MODE <mode>

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

The :MARKer:MODE command sets the cursors mode:

- OFF – removes the cursor information from the display.
- MANual – enables manual placement of the X and Y cursors.

  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.
- MEASurement – cursors track the most recent measurement.

  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.
- WAVeform – the Y1 cursor tracks the voltage value at the X1 cursor of the waveform specified by the X1Y1source, and the Y2 cursor does the same for the X2 cursor and its X2Y2source.
- BINary – logic levels of displayed waveforms at the current X1 and X2 cursor positions are displayed in the Cursor sidebar dialog in binary.
- HEX – logic levels of displayed waveforms at the current X1 and X2 cursor positions are displayed in the Cursor sidebar dialog in hexadecimal.

Query Syntax

```
:MARKer:MODE?
```

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

Return Format

```
<mode><NL>

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

See Also

- "Introduction to :MARKer Commands" on page 509
- ":MARKer:X1Y1source" on page 514
- ":MARKer:X2Y2source" on page 517
- ":MEASure:SOURce" on page 599
- ":MARKer:X1Position" on page 513
- ":MARKer:X2Position" on page 516
- ":MARKer:Y1Position" on page 522
- ":MARKer:Y2Position" on page 524

:MARKer:X1:DISPlay

 (see page 1292)

Command Syntax `:MARKer:X1:DISPlay {{0 | OFF} | {1 | ON}}`

The :MARKer:X1:DISPlay command specifies whether the X1 cursor is displayed.

Query Syntax `:MARKer:X1:DISPlay?`

The :MARKer:X1:DISPlay? query returns the X1 cursor display setting.

Return Format `<setting><NL>`

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

See Also

- ":MARKer:X1:DISPlay" on page 512

:MARKer:X1Position

(see page 1292)

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 if it is not currently set to WAVeform (see ":MARKer:MODE" on page 511).
- Sets the X1 cursor position to the specified value.

X cursor units are set by the :MARKer:XUNits command.

Query Syntax

```
:MARKer:X1Position?
```

The :MARKer:X1Position? query returns the current X1 cursor position. This is functionally equivalent to the obsolete :MEASure:TSTArt command/query.

NOTE

If the front-panel cursors are off, the marker position values are not defined and an error is generated. Make sure to set :MARKer:MODE to MANual or WAVeform to put the cursors in the front-panel Normal mode.

Return Format

```
<position><NL>

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

See Also

- "Introduction to :MARKer Commands" on page 509
- ":MARKer:MODE" on page 511
- ":MARKer:X2Position" on page 516
- ":MARKer:X1Y1source" on page 514
- ":MARKer:X2Y2source" on page 517
- ":MARKer:XUNits" on page 519

:MARKer:X1Y1source

N (see page 1292)

Command Syntax

```
:MARKer:X1Y1source <source>

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | FFT | WMEMory<r>
   | HISTogram}

<n> ::= 1 to (# analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

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.

If the marker mode is not currently WAVeform (see ":MARKer:MODE" on page 511):

- Sending a :MARKer:X1Y1source command will put the cursors in the MANual mode.
- Setting the source for one pair of markers (for example, X1Y1) sets the source for the other (for example, X2Y2).

If the marker mode is currently WAVeform, the X1Y1 source can be set separate from the X2Y2 source.

If :MARKer:MODE is set to OFF or MANual, setting :MEASure:SOURce to CHANnel<n>, FUNCtion, MATH, or WMEMory<r> 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 | WMEM<r> | HIST | NONE}
```

See Also

- "Introduction to :MARKer Commands" on page 509
- ":MARKer:MODE" on page 511
- ":MARKer:X2Y2source" on page 517
- ":MEASure:SOURce" on page 599

:MARKer:X2:DISPlay

(see page 1292)

Command Syntax `:MARKer:X2:DISPlay {{0 | OFF} | {1 | ON}}`

The :MARKer:X2:DISPlay command specifies whether the X2 cursor is displayed.

Query Syntax `:MARKer:X2:DISPlay?`

The :MARKer:X2:DISPlay? query returns the X2 cursor display setting.

Return Format `<setting><NL>`

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

See Also

- ":MARKer:X2:DISPlay" on page 515

:MARKer:X2Position

(see page 1292)

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 if it is not currently set to WAVeform (see ":MARKer:MODE" on page 511).
- Sets the X2 cursor position to the specified value.

X cursor units are set by the :MARKer:XUNits command.

Query Syntax

```
:MARKer:X2Position?
```

The :MARKer:X2Position? query returns current X2 cursor position. This is functionally equivalent to the obsolete :MEASure:TSTOp command/query.

NOTE If the front-panel cursors are off, the marker position values are not defined and an error is generated. Make sure to set :MARKer:MODE to MANual or WAVeform to put the cursors in the front-panel Normal mode.

Return Format

```
<position><NL>

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

See Also

- "Introduction to :MARKer Commands" on page 509
- ":MARKer:MODE" on page 511
- ":MARKer:X1Position" on page 513
- ":MARKer:X2Y2source" on page 517
- ":MARKer:XUNits" on page 519

:MARKer:X2Y2source

(see page 1292)

Command Syntax

```
:MARKer:X2Y2source <source>

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | FFT | WMEMory<r>
   | HISTogram}

<n> ::= 1 to (# analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

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.

If the marker mode is not currently WAVeform (see ":MARKer:MODE" on page 511):

- Sending a :MARKer:X2Y2source command will put the cursors in the MANual mode.
- Setting the source for one pair of markers (for example, X2Y2) sets the source for the other (for example, X1Y1).

If the marker mode is currently WAVeform, the X2Y2 source can be set separate from the X1Y1 source.

If :MARKer:MODE is set to OFF or MANual, setting :MEASure:SOURce to CHANnel<n>, FUNCtion, MATH, or WMEMory<r> 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 | WMEM<r> | HIST | NONE}
```

See Also

- "Introduction to :MARKer Commands" on page 509
- ":MARKer:MODE" on page 511
- ":MARKer:X1Y1source" on page 514
- ":MEASure:SOURce" on page 599

:MARKer:XDELta?

(see page 1292)

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)

X cursor units are set by the :MARKer:XUNits command.

NOTE If the front-panel cursors are off, the marker position values are not defined. Make sure to set :MARKer:MODE to MANual or WAVeform 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" on page 509
- ":MARKer:MODE" on page 511
- ":MARKer:X1Position" on page 513
- ":MARKer:X2Position" on page 516
- ":MARKer:X1Y1source" on page 514
- ":MARKer:X2Y2source" on page 517
- ":MARKer:XUNits" on page 519

:MARKer:XUNits

(see page 1292)

Command Syntax

```
:MARKer:XUNits <units>

<units> ::= {SEConds | HERTz | DEGRees | PERCent}
```

The :MARKer:XUNits command sets the X cursors units:

- SEConds – for making time measurements.
- HERTz – for making frequency measurements.
- DEGRees – for making phase measurements. Use the :MARKer:XUNits:USE command to set the current X1 location as 0 degrees and the current X2 location as 360 degrees.
- PERCent – for making ratio measurements. Use the :MARKer:XUNits:USE command to set the current X1 location as 0 percent and the current X2 location as 100 percent.

Changing X units affects the input and output values of the :MARKer:X1Position, :MARKer:X2Position, and :MARKer:XDELta commands/queries.

Query Syntax

```
:MARKer:XUNits?
```

The :MARKer:XUNits? query returns the current X cursors units.

Return Format

```
<units><NL>

<units> ::= {SEC | HERT | DEGR | PERC}
```

See Also

- "Introduction to :MARKer Commands" on page 509
- ":MARKer:XUNits:USE" on page 520
- ":MARKer:X1Y1source" on page 514
- ":MARKer:X2Y2source" on page 517
- ":MEASure:SOURce" on page 599
- ":MARKer:X1Position" on page 513
- ":MARKer:X2Position" on page 516

:MARKer:XUNits:USE

(see page 1292)

Command Syntax `:MARKer:XUNits:USE`

When DEGRees is selected for :MARKer:XUNits, the :MARKer:XUNits:USE command sets the current X1 location as 0 degrees and the current X2 location as 360 degrees.

When PERCent is selected for :MARKer:XUNits, the :MARKer:XUNits:USE command sets the current X1 location as 0 percent and the current X2 location as 100 percent.

Once the 0 and 360 degree or 0 and 100 percent locations are set, inputs to and outputs from the :MARKer:X1Position, :MARKer:X2Position, and :MARKer:XDELta commands/queries are relative to the set locations.

See Also

- "Introduction to :MARKer Commands" on page 509
- ":MARKer:XUNits" on page 519
- ":MARKer:X1Y1source" on page 514
- ":MARKer:X2Y2source" on page 517
- ":MEASure:SOURce" on page 599
- ":MARKer:X1Position" on page 513
- ":MARKer:X2Position" on page 516
- ":MARKer:XDELta?" on page 518

:MARKer:Y1:DISPlay

N (see page 1292)

Command Syntax `:MARKer:Y1:DISPlay {{0 | OFF} | {1 | ON}}`

The :MARKer:Y1:DISPlay command specifies whether the Y1 cursor is displayed.

Query Syntax `:MARKer:Y1:DISPlay?`

The :MARKer:Y1:DISPlay? query returns the Y1 cursor display setting.

Return Format `<setting><NL>`

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

See Also

- ":MARKer:Y1:DISPlay" on page 521

:MARKer:Y1Position

(see page 1292)

Command Syntax

```
:MARKer:Y1Position <position> [suffix]

<position> ::= Y1 cursor position in NR3 format

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

If the :MARKer:MODE is not currently set to WAVeform (see ":MARKer:MODE" on page 511), the :MARKer:Y1Position command:

- Sets :MARKer:MODE to MANual.
- Sets the Y1 cursor position to the specified value.

Y cursor units are set by the :MARKer:YUNits command.

When the :MARKer:MODE is set to WAVeform, Y positions cannot be set.

Query Syntax

```
:MARKer:Y1Position?
```

The :MARKer:Y1Position? query returns current Y1 cursor position. This is functionally equivalent to the obsolete :MEASure:VSTArt command/query.

NOTE If the front-panel cursors are off or are set to Binary or Hex Mode, the marker position values are not defined and an error is generated. Make sure to set :MARKer:MODE to MANual or WAVeform to put the cursors in the front-panel Normal mode.

Return Format

```
<position><NL>

<position> ::= Y1 cursor position in NR3 format
```

See Also

- "Introduction to :MARKer Commands" on page 509
- ":MARKer:MODE" on page 511
- ":MARKer:X1Y1source" on page 514
- ":MARKer:X2Y2source" on page 517
- ":MARKer:Y2Position" on page 524
- ":MARKer:YUNits" on page 526

:MARKer:Y2:DISPlay

N (see page 1292)

Command Syntax `:MARKer:Y2:DISPlay {{0 | OFF} | {1 | ON}}`

The :MARKer:Y2:DISPlay command specifies whether the Y2 cursor is displayed.

Query Syntax `:MARKer:Y2:DISPlay?`

The :MARKer:Y2:DISPlay? query returns the Y2 cursor display setting.

Return Format `<setting><NL>`

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

See Also

- ":MARKer:Y2:DISPlay" on page 523

:MARKer:Y2Position

(see page 1292)

Command Syntax

```
:MARKer:Y2Position <position> [suffix]

<position> ::= Y2 cursor position in NR3 format

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

If the :MARKer:MODE is not currently set to WAVeform (see ":MARKer:MODE" on page 511), the :MARKer:Y1Position command:

- Sets :MARKer:MODE to MANual.
- Sets the Y2 cursor position to the specified value.

Y cursor units are set by the :MARKer:YUNits command.

When the :MARKer:MODE is set to WAVeform, Y positions cannot be set.

Query Syntax

```
:MARKer:Y2Position?
```

The :MARKer:Y2Position? query returns current Y2 cursor position. This is functionally equivalent to the obsolete :MEASure:VSTOp command/query.

NOTE If the front-panel cursors are off or are set to Binary or Hex Mode, the marker position values are not defined and an error is generated. Make sure to set :MARKer:MODE to MANual or WAVeform to put the cursors in the front-panel Normal mode.

Return Format

```
<position><NL>

<position> ::= Y2 cursor position in NR3 format
```

See Also

- "Introduction to :MARKer Commands" on page 509
- ":MARKer:MODE" on page 511
- ":MARKer:X1Y1source" on page 514
- ":MARKer:X2Y2source" on page 517
- ":MARKer:Y1Position" on page 522
- ":MARKer:YUNits" on page 526

:MARKer:YDELta?

(see page 1292)

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 or WAVeform to put the cursors in the front-panel Normal mode.**

Y cursor units are set by the :MARKer:YUNits command.

Return Format `<value><NL>`

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

See Also

- "Introduction to :MARKer Commands" on page 509
- ":MARKer:MODE" on page 511
- ":MARKer:X1Y1source" on page 514
- ":MARKer:X2Y2source" on page 517
- ":MARKer:Y1Position" on page 522
- ":MARKer:Y2Position" on page 524
- ":MARKer:YUNits" on page 526

:MARKer:YUNits

(see page 1292)

Command Syntax

```
:MARKer:YUNits <units>

<units> ::= {BASE | PERCent}
```

The :MARKer:YUNits command sets the Y cursors units:

- BASE – for making measurements in the units associated with the cursors source.
- PERCent – for making ratio measurements. Use the :MARKer:YUNits:USE command to set the current Y1 location as 0 percent and the current Y2 location as 100 percent.

Changing Y units affects the input and output values of the :MARKer:Y1Position, :MARKer:Y2Position, and :MARKer:YDELta commands/queries.

Query Syntax

```
:MARKer:YUNits?
```

The :MARKer:YUNits? query returns the current Y cursors units.

Return Format

```
<units><NL>

<units> ::= {BASE | PERC}
```

See Also

- "Introduction to :MARKer Commands" on page 509
- ":MARKer:YUNits:USE" on page 527
- ":MARKer:X1Y1source" on page 514
- ":MARKer:X2Y2source" on page 517
- ":MEASure:SOURce" on page 599
- ":MARKer:Y1Position" on page 522
- ":MARKer:Y2Position" on page 524
- ":MARKer:YDELta?" on page 525

:MARKer:YUNits:USE

(see page 1292)

Command Syntax `:MARKer:YUNits:USE`

When PERCent is selected for :MARKer:YUNits, the :MARKer:YUNits:USE command sets the current Y1 location as 0 percent and the current Y2 location as 100 percent.

Once the 0 and 100 percent locations are set, inputs to and outputs from the :MARKer:Y1Position, :MARKer:Y2Position, and :MARKer:YDELta commands/queries are relative to the set locations.

See Also

- "Introduction to :MARKer Commands" on page 509
- ":MARKer:YUNits" on page 526
- ":MARKer:X1Y1source" on page 514
- ":MARKer:X2Y2source" on page 517
- ":MEASure:SOURce" on page 599
- ":MARKer:Y1Position" on page 522
- ":MARKer:Y2Position" on page 524
- ":MARKer:YDELta?" on page 525

25 :MEASure Commands

Select automatic measurements to be made and control time markers. See "Introduction to :MEASure Commands" on page 545.

Table 90 :MEASure Commands Summary

Command	Query	Options and Query Returns
:MEASure:AREa [<interval>][,<source>] (see page 547)	:MEASure:AREa? [<interval>][,<source>] (see page 547)	<interval> ::= {CYCLe | DISPlay} <source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= area in volt-seconds, NR3 format
:MEASure:BRATe [<source>] (see page 548)	:MEASure:BRATe? [<source>] (see page 548)	<source> ::= {<digital channels> | CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <digital channels> ::= DIGital<d> <d> ::= 0 to (# digital channels - 1) in NR1 format <n> ::= 1 to (# of analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= bit rate in Hz, NR3 format

Table 90 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:BWIDth [<source>] (see page 549)	:MEASure:BWIDth? [<source>] (see page 549)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= burst width in seconds, NR3 format
:MEASure:CLEar (see page 550)	n/a	n/a
:MEASure:COUNter [<source>] (see page 551)	:MEASure:COUNter? [<source>] (see page 551)	<source> ::= {CHANnel<n> | DIGital<d> | EXTernal} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= counter frequency in Hertz in NR3 format
:MEASure:DELay [<source1>] [,<source2>] (see page 553)	:MEASure:DELay? [<source1>] [,<source2>] (see page 553)	<source1,2> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r> | <digital channels>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <digital channels> ::= DIGital<d> <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= floating-point number delay time in seconds in NR3 format

Table 90 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:DELay:DEFine <source1_edge_slope>, <source1_edge_number>, <source1_edge_threshold>, <source2_edge_slope>, <source2_edge_number>, <source2_edge_threshold> (see page 555)	:MEASure:DELay:DEFine? (see page 555)	<source1_edge_slope>, <source2_edge_slope> ::= {RISing | FALLing} <source1_edge_number>, <source2_edge_number> ::= 0 to 1000 in NR1 format <source1_edge_threshold>, <source2_edge_threshold> ::= {LOWer | MIDDle | UPPer}
:MEASure:DUTYcycle [<source>] (see page 557)	:MEASure:DUTYcycle? [<source>] (see page 557)	<source> ::= {CHANnel<n> | DIGital<d> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= ratio of positive pulse width to period in NR3 format
:MEASure:FALLtime [<source>] (see page 558)	:MEASure:FALLtime? [<source>] (see page 558)	<source> ::= {CHANnel<n> | DIGital<d> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= time in seconds between the lower and upper thresholds in NR3 format

Table 90 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:FFT:ACPR <chan_width>, <chan_spacing>, <chan>[,<source>] (see page 559)	:MEASure:FFT:ACPR? <chan_width>, <chan_spacing>, <chan>[,<source>] (see page 559)	<chan_width> ::= width of main range and sideband channels, Hz in NR3 format <chan_spacing> ::= spacing between main range and sideband channels, Hz in NR3 format <chan> ::= {CENTer | HIGH<sb> | LOW<sb>} <sb> ::= sideband 1 to 5 <source> ::= {FUNCtion<m> | MATH<m> | FFT} (source must be an FFT waveform) <m> ::= 1 to (# math functions) in NR1 format <return_value> ::= adjacent channel power ratio, dBV in NR3 format
:MEASure:FFT:CPOWer [<source>] (see page 560)	:MEASure:FFT:CPOWer? [<source>] (see page 560)	<source> ::= {FUNCtion<m> | MATH<m> | FFT} (source must be an FFT waveform) <m> ::= 1 to (# math functions) in NR1 format <return_value> ::= spectral channel power, dBV in NR3 format
:MEASure:FFT:OBW <percentage>[,<source>] (see page 561)	:MEASure:FFT:OBW? <percentage>[,<source>] (see page 561)	<percentage> ::= percent of spectral power occupied bandwidth is measured for in NR3 format <source> ::= {FUNCtion<m> | MATH<m> | FFT} (source must be an FFT waveform) <m> ::= 1 to (# math functions) in NR1 format <return_value> ::= occupied bandwidth, Hz in NR3 format

Table 90 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:FFT:THD [<source>] (see page 562)	:MEASure:FFT:THD? [<source>] (see page 562)	<source> ::= {FUNCtion<m> | MATH<m> | FFT} (source must be an FFT waveform) <m> ::= 1 to (# math functions) in NR1 format <return_value> ::= total harmonic distortion ratio percent in NR3 format
:MEASure:FREQuency [<source>] (see page 563)	:MEASure:FREQuency? [<source>] (see page 563)	<source> ::= {CHANnel<n> | DIGital<d> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= frequency in Hertz in NR3 format
n/a	:MEASure:HISTogram:BWIDth? (see page 564)	<value> ::= bin width in NR3 format
n/a	:MEASure:HISTogram:HITS? (see page 565)	<value> ::= # of total hits in NR3 format
n/a	:MEASure:HISTogram:M1S? (see page 566)	<value> ::= % of hits within +/-1 std dev in NR3 format
n/a	:MEASure:HISTogram:M2S? (see page 567)	<value> ::= % of hits within +/-2 std devs in NR3 format
n/a	:MEASure:HISTogram:M3S? (see page 568)	<value> ::= % of hits within +/-3 std devs in NR3 format
n/a	:MEASure:HISTogram:MAXimum? (see page 569)	<value> ::= value of max bin in NR3 format
n/a	:MEASure:HISTogram:MEAN? (see page 570)	<value> ::= mean of histogram in NR3 format
n/a	:MEASure:HISTogram:MEDian? (see page 571)	<value> ::= median of histogram in NR3 format
n/a	:MEASure:HISTogram:MINimum? (see page 572)	<value> ::= value of min bin in NR3 format

Table 90 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:MEASure:HISTogram:MODE? (see page 573)	<value> ::= mode of histogram in NR3 format
n/a	:MEASure:HISTogram:PEAK? (see page 574)	<value> ::= # of hits in max bin in NR3 format
n/a	:MEASure:HISTogram:PPEak? (see page 575)	<value> ::= delta value between max bin and min bin in NR3 format
n/a	:MEASure:HISTogram:SDEViation? (see page 576)	<value> ::= standard deviation of histogram in NR3 format
:MEASure:NDUTy [<source>] (see page 577)	:MEASure:NDUTy? [<source>] (see page 577)	<source> ::= {CHANnel<n> | DIGital<d> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= ratio of negative pulse width to period in NR3 format
:MEASure:NEDGes [<source>] (see page 578)	:MEASure:NEDGes? [<source>] (see page 578)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the falling edge count in NR3 format

Table 90 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:NPULses [<source>] (see page 579)	:MEASure:NPULses? [<source>] (see page 579)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r> | <digital channels>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <digital channels> ::= DIGital<d> <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= the falling pulse count in NR3 format
:MEASure:NWIDth [<source>] (see page 580)	:MEASure:NWIDth? [<source>] (see page 580)	<source> ::= {CHANnel<n> | DIGital<d> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= negative pulse width in seconds-NR3 format
:MEASure:OVERshoot [<source>] (see page 581)	:MEASure:OVERshoot? [<source>] (see page 581)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the percent of the overshoot of the selected waveform in NR3 format

Table 90 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:PEDGes [<source>] (see page 583)	:MEASure:PEDGes? [<source>] (see page 583)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the rising edge count in NR3 format
:MEASure:PERiod [<source>] (see page 584)	:MEASure:PERiod? [<source>] (see page 584)	<source> ::= {CHANnel<n> | DIGital<d> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= waveform period in seconds in NR3 format
:MEASure:PHASe [<source1>] [,<source2>] (see page 585)	:MEASure:PHASe? [<source1>] [,<source2>] (see page 585)	<source1,2> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the phase angle value in degrees in NR3 format

Table 90 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:PPULses [<source>] (see page 586)	:MEASure:PPULses? [<source>] (see page 586)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r> | <digital channels>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <digital channels> ::= DIGital<d> <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= the rising pulse count in NR3 format
:MEASure:PREShoot [<source>] (see page 587)	:MEASure:PREShoot? [<source>] (see page 587)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the percent of preshoot of the selected waveform in NR3 format
:MEASure:PWIDth [<source>] (see page 588)	:MEASure:PWIDth? [<source>] (see page 588)	<source> ::= {CHANnel<n> | DIGital<d> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= width of positive pulse in seconds in NR3 format

Table 90 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:QUICk <source> (see page 589)	n/a	<source> ::= {CHANnel<n>}
n/a	:MEASure:RESults? <result_list> (see page 590)	<result_list> ::= comma-separated list of measurement results
:MEASure:RISetime [<source>] (see page 594)	:MEASure:RISetime? [<source>] (see page 594)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= rise time in seconds in NR3 format
:MEASure:SDEViation [<source>] (see page 595)	:MEASure:SDEViation? [<source>] (see page 595)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= calculated std deviation in NR3 format
:MEASure:SHOW {{0 | OFF} | {1 | ON}} (see page 596)	:MEASure:SHOW? (see page 596)	{0 | 1}
:MEASure:SLEWrate [<source>[,<slope]] (see page 597)	:MEASure:SLEWrate? [<source>[,<slope]] (see page 597)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# of analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format

Table 90 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:SOURce <source1> [,<source2>] (see page 599)	:MEASure:SOURce? (see page 599)	<source1,2> ::= {CHANnel<n> | DIGital<d> | FUNCtion<m> | MATH<m> | WMEMory<r> | EXTernal} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= {<source> | NONE}
:MEASure:STATistics <type> (see page 602)	:MEASure:STATistics? (see page 602)	<type> ::= {{ON | 1} | CURRent | MEAN | MINimum | MAXimum | STDDev | COUNt} ON ::= all statistics returned
:MEASure:STATistics:DISPlay {{0 | OFF} | {1 | ON}} (see page 603)	:MEASure:STATistics:DISPlay? (see page 603)	{0 | 1}
:MEASure:STATistics:INCRement (see page 604)	n/a	n/a
:MEASure:STATistics:MCOunt <setting> (see page 605)	:MEASure:STATistics:MCOunt? (see page 605)	<setting> ::= {INFinite | <count>} <count> ::= 2 to 2000 in NR1 format
:MEASure:STATistics:RESet (see page 606)	n/a	n/a
:MEASure:STATistics:RSDeviation {{0 | OFF} | {1 | ON}} (see page 607)	:MEASure:STATistics:RSDeviation? (see page 607)	{0 | 1}

Table 90 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:TEDGe [<slope>,]<occurrence>[,<source>] (see page 608)	:MEASure:TEDGe? [<slope>,]<occurrence>[,<source>] (see page 609)	<slope> ::= {RISing | FALLing | BOTH} <occurrence> ::= [+ | -]<number> <number> ::= the edge number in NR1 format <source> ::= {<digital channels> | CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <digital channels> ::= DIGital<d> <n> ::= 1 to (# of analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= time in seconds of the specified transition
:MEASure:THResholds:ABSolute <upper>,<middle>,<lower> (see page 612)	:MEASure:THResholds:ABSolute? (see page 612)	<upper>,<middle>,<lower> ::= A number specifying the upper, middle, and lower threshold absolute values in NR3 format.
:MEASure:THResholds:METHod <threshold_mode> (see page 613)	:MEASure:THResholds:METHod? (see page 613)	<threshold_mode> ::= {PERCent | ABSolute}
:MEASure:THResholds:PERCent <upper>,<middle>,<lower> (see page 614)	:MEASure:THResholds:PERCent? (see page 614)	<upper>,<middle>,<lower> ::= A number specifying the upper, middle, and lower threshold percentage values between Vbase and Vtop in NR3 format.
:MEASure:THResholds:STANdard (see page 615)	n/a	n/a

Table 90 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:MEASure:TVALue? <value>, [<slope>]<occurrence> [,<source>] (see page 616)	<value> ::= voltage level that the waveform must cross. <slope> ::= direction of the waveform when <value> is crossed. <occurrence> ::= transitions reported. <source> ::= {CHANnel<n> | DIGital<d> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= time in seconds of specified voltage crossing in NR3 format
:MEASure:VAMPlitude [<source>] (see page 618)	:MEASure:VAMPlitude? [<source>] (see page 618)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the amplitude of the selected waveform in volts in NR3 format

Table 90 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:VAVerage [<interval>][,<source>] (see page 619)	:MEASure:VAVerage? [<interval>][,<source>] (see page 619)	<interval> ::= {CYCLe | DISPlay} <source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | FFT | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= calculated average voltage in NR3 format
:MEASure:VBASe [<source>] (see page 620)	:MEASure:VBASe? [<source>] (see page 620)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <base_voltage> ::= voltage at the base of the selected waveform in NR3 format
:MEASure:VMAX [<source>] (see page 621)	:MEASure:VMAX? [<source>] (see page 621)	<source> ::= {CHANnel<n> | FUNCtion<m> | FFT | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= maximum voltage of the selected waveform in NR3 format

Table 90 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:VMIN [<source>] (see page 622)	:MEASure:VMIN? [<source>] (see page 622)	<source> ::= {CHANnel<n> | FUNCtion<m> | FFT | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= minimum voltage of the selected waveform in NR3 format
:MEASure:VPP [<source>] (see page 623)	:MEASure:VPP? [<source>] (see page 623)	<source> ::= {CHANnel<n> | FUNCtion<m> | FFT | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= voltage peak-to-peak of the selected waveform in NR3 format
:MEASure:VRATio [<interval>][,<source 1>] [,<source2>] (see page 624)	:MEASure:VRATio? [<interval>][,<source 1>] [,<source2>] (see page 624)	<interval> ::= {CYCLe | DISPlay} <source1,2> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the ratio value in dB in NR3 format

Table 90 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:VRMS [<interval>] [,<type>] [,<source>] (see page 625)	:MEASure:VRMS? [<interval>] [,<type>] [,<source>] (see page 625)	<interval> ::= {CYCLe | DISPlay} <type> ::= {AC | DC} <source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= calculated dc RMS voltage in NR3 format
:MEASure:VTOP [<source>] (see page 626)	:MEASure:VTOP? [<source>] (see page 626)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= voltage at the top of the waveform in NR3 format
:MEASure:WINDow <type> (see page 627)	:MEASure:WINDow? (see page 627)	<type> ::= {MAIN | ZOOM | AUTO | GATE}
:MEASure:XMAX [<source>] (see page 628)	:MEASure:XMAX? [<source>] (see page 628)	<source> ::= {CHANnel<n> | FUNCtion<m> | FFT | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= horizontal value of the maximum in NR3 format

Table 90 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:XMIN [<source>] (see page 629)	:MEASure:XMIN? [<source>] (see page 629)	<source> ::= {CHANnel<n> | FUNCtion<m> | FFT | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= horizontal value of the minimum in NR3 format
:MEASure:YATX <horiz_location>[,<source>] (see page 630)	:MEASure:YATX? <horiz_location>[,<source>] (see page 630)	<horiz_location> ::= displayed time from trigger in seconds in NR3 format <source> ::= {CHANnel<n> | DIGital<d> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= voltage at the specified time 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

Measurement Type	Portion of waveform that must be displayed
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 zoomed (delayed) time base mode (:TIMebase:MODE WINDow), the oscilloscope will attempt to make the measurement inside the zoomed 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,CHAN2;STAT ON
```

:MEASure:AREA

(see page 1292)

Command Syntax

```
:MEASure:AREA [<interval>][,<source>]

<interval> ::= {CYCLe | DISPlay}

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

<n> ::= 1 to (# analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :MEASure:AREA command installs an area measurement on screen. Area measurements show the area between the waveform and the ground level.

The <interval> option lets you specify the measurement interval: either an integral number of cycles, or the full screen. If <interval> is not specified, DISPlay is implied.

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:AREA? [<interval>][,<source>]
```

The :MEASure:AREA? query measures and returns the area value.

Return Format

```
<value><NL>

<value> ::= the area value in volt-seconds in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 545
- ":MEASure:SOURce" on page 599

:MEASure:BRATe

(see page 1292)

Command Syntax

```
:MEASure:BRATe [<source>]

<source> ::= {<digital channels> | CHANnel<n> | FUNCtion<m> | MATH<m>
   | WMEMory<r>}

<digital channels> ::= DIGital<d>

<d> ::= 0 to (# digital channels - 1) in NR1 format

<n> ::= 1 to (# of analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :MEASure:BRATe command installs a screen measurement and starts the bit rate measurement. If the optional source parameter is specified, the currently specified source is modified.

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:BRATe? [<source>]
```

The :MEASure:BRATe? query measures all positive and negative pulse widths on the waveform, takes the minimum value found of either width type and inverts that minimum width to give a value in Hertz.

Return Format

```
<value><NL>

<value> ::= the bit rate value in Hertz
```

See Also

- "Introduction to :MEASure Commands" on page 545
- ":MEASure:SOURce" on page 599
- ":MEASure:FREQuency" on page 563
- ":MEASure:PERiod" on page 584

:MEASure:BWIDth

N (see page 1292)

Command Syntax

```
:MEASure:BWIDth [<source>]

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

<n> ::= 1 to (# analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :MEASure:BWIDth command installs a burst width measurement on screen. If the optional source parameter is not specified, the current measurement source is used.

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:BWIDth? [<source>]
```

The :MEASure:BWIDth? query measures and returns the width of the burst on the screen.

The burst width is calculated as follows:

burst width = (last edge on screen - first edge on screen)

Return Format

```
<value><NL>

<value> ::= burst width in seconds in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 545
- ":MEASure:SOURce" on page 599

:MEASure:CLEar

 (see page 1292)

Command Syntax `:MEASure:CLEar`

This command clears all selected measurements and markers from the screen.

See Also

- "Introduction to :MEASure Commands" on page 545

:MEASure:COUNter

(see page 1292)

Command Syntax

```
:MEASure:COUNter [<source>]

<source> ::= {<digital channels> | CHANnel<n> | EXTernal}

<digital channels> ::= DIGital<d>

<n> ::= 1 to (# analog channels) in NR1 format

<d> ::= 0 to (# digital channels - 1) in NR1 format
```

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 or Reference Waveforms may be selected for the source.

The counter measurement counts trigger level crossings within a certain amount of time (gate time) and displays the results in Hz.

The gate time is the horizontal range of the oscilloscope but is limited to >= 0.1 s and <= 10 s. Unlike other measurements, the Zoom horizontal timebase window does not gate the Counter measurement.

The Counter measurement can measure frequencies up to the bandwidth of the oscilloscope. The minimum frequency supported is 2.0 / gateTime.

Only one counter measurement may be displayed at a time.

NOTE This command is not available if the source is MATH.

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" on page 545
- ":MEASure:SOURce" on page 599

- ":MEASure:FREQuency" on page 563
- ":MEASure:CLEar" on page 550

:MEASure:DELay

N (see page 1292)

Command Syntax

```
:MEASure:DELay [<edge_select_mode>][,][<source1>][,<source2>]

<edge_select_mode> ::= {MANual | AUTO}

<source1>, <source2> ::= {CHANnel<n> | FUNCtion<m> | MATH<m>
   | WMEMory<r> | <digital channels>}

<n> ::= 1 to (# analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format

<digital channels> ::= DIGital<d>

<d> ::= 0 to (# digital channels - 1) in NR1 format
```

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 determined by the <edge_select_mode> and the :MEASure:DELay:DEFine command.

When the <edge_select_mode> is:

- AUTO – edges are automatically selected: the source1 edge closest to the timebase reference point is used, and the source2 edge closest to the source1 edge is used.
- MANual – edge numbers are determined by the :MEASure:DELay:DEFine settings.

  Edge numbers greater than zero (0) are counted from the left side of the display for both sources.

Edge thresholds and slopes are always determined by the :MEASure:DELay:DEFine settings.

If the <edge_select_mode> is not included with the command, AUTO is the default.

Query Syntax

```
:MEASure:DELay? [<edge_select_mode>][,][<source1>][,<source2>]
```

The :MEASure:DELay? query measures and returns the delay between source1 and source2. Delay measurement threshold, slope, and edge count parameters are determined by the <edge_select_mode> and the :MEASure:DELay:DEFine command.

The <edge_select_mode> definitions are the same as with the command syntax.

However, if the <edge_select_mode> is not included with the query, MANual is the default.

In the :MEASure:DEFine command, you can set upper, middle, and lower threshold values. The standard upper, middle, and lower measurement thresholds are 90%, 50%, and 10% values between Vbase and Vtop.

Return Format

```
<value><NL>

<value> ::= floating-point number delay time in seconds in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 545
- ":MEASure:DELay:DEFine" on page 555
- ":MEASure:DELay:DEFine" on page 555
- ":MEASure:THResholds:ABSolute" on page 612
- ":MEASure:THResholds:METHod" on page 613
- ":MEASure:THResholds:PERCent" on page 614
- ":MEASure:THResholds:STANdard" on page 615
- ":MEASure:PHASe" on page 585

:MEASure:DELay:DEFine

N (see page 1292)

Command Syntax

```
:MEASure:DELay:DEFine <source1_edge_slope>, <source1_edge_number>,
   <source1_edge_threshold>, <source2_edge_slope>,
   <source2_edge_number>, <source2_edge_threshold>

<source1_edge_slope>,
<source2_edge_slope> ::= {RISing | FALLing}

<source1_edge_number>,
<source2_edge_number> ::= 0 to 1000 in NR1 format

<source1_edge_threshold>,
<source2_edge_threshold> ::= {LOWer | MIDDle | UPPer}
```

The :MEASure:DELay:DEFine command defines slope directions and edge numbers for the delay measurement started or returned by the :MEASure:DELay command. The :MEASure:DELay:DEFine command also defines edge position parameters.

A <source1_edge_number> setting of zero (0) specifies that the edge closest to the timebase reference point automatically be selected. In this case, the <source2_edge_number> setting must also be zero (0), and the source2 edge closest to the selected source1 edge is used.

When edge numbers greater than zero (0) are specified, edges are counted from the left side of the display for both sources.

The selection of the source1 and source2 waveforms is made in the :MEASure:DELay or :MEASure:SOURce commands.

Query Syntax

```
:MEASure:DELay:DEFine?
```

The :MEASure:DELay:DEFine? query returns the specified delay measurement parameter definitions.

Return Format

```
<source1_edge_slope>, <source1_edge_number>,
   <source1_edge_threshold>, <source2_edge_slope>,
   <source2_edge_number>,<source2_edge_threshold><NL>

<source1_edge_slope>,
<source2_edge_slope> ::= {RIS | FALL}

<source1_edge_number>,
<source2_edge_number> ::= 0 to 1000 in NR1 format

<source1_edge_threshold>,
<source2_edge_threshold> ::= {LOW | MIDD | UPP}
```

See Also

- ":MEASure:DELay" on page 553
- ":MEASure:SOURce" on page 599
- ":MEASure:THResholds:ABSolute" on page 612

- ":MEASure:THResholds:METHod" on page 613
- ":MEASure:THResholds:PERCent" on page 614
- ":MEASure:THResholds:STANdard" on page 615

:MEASure:DUTYcycle

(see page 1292)

Command Syntax

```
:MEASure:DUTYcycle [<source>]

<source> ::= {<digital channels> | CHANnel<n> | FUNCtion<m> | MATH<m>
             | WMEMory<r>}

<digital channels> ::= DIGital<d>

<d> ::= 0 to (# digital channels - 1) in NR1 format

<n> ::= 1 to (# of analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :MEASure:DUTYcycle command installs a screen measurement and starts a positive 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 positive 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 in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 545
- ":MEASure:PERiod" on page 584
- ":MEASure:PWIDth" on page 588
- ":MEASure:SOURce" on page 599
- ":MEASure:NDUTy" on page 577

Example Code

- "Example Code" on page 600

:MEASure:FALLtime

(see page 1292)

Command Syntax

```
:MEASure:FALLtime [<source>]

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

<n> ::= 1 to (# analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

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 timebase 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" on page 545
- ":MEASure:RISetime" on page 594
- ":MEASure:SOURce" on page 599
- ":MEASure:SLEWrate" on page 597

:MEASure:FFT:ACPR

N (see page 1292)

Command Syntax

```
:MEASure:FFT:ACPR <chan_width>,<chan_spacing>,<chan>[,<source>]

<chan_width> ::= of main range and sideband channels, Hz in NR3 format

<chan_spacing> ::= spacing between main range and sideband channels,
   Hz in NR3 format

<chan> ::= {CENTer | HIGH<sb> | LOW<sb>}

<sb> ::= sideband 1 to 5

<source> ::= {FUNCtion<m> | MATH<m> | FFT} (must be an FFT waveform)

<m> ::= 1 to (# math functions) in NR1 format
```

The :MEASure:FFT:ACPR command installs an FFT analysis Adjacent Channel Power Ratio (ACPR) measurement on screen.

Adjacent Channel Power Ratio (or channel leakage ratio) measures the ratio of the power in the main frequency range to the power contained in one or more sidebands.

The main range is specified by a channel width and a center frequency. The center frequency used in the measurement is the one defined for the FFT function.

Sidebands (with the same width as the main range) exist above and below the main range separated by the channel spacing width.

The sideband used for the measurement is selected with the <chan> parameter. You can select the first through fifth sidebands above or below the main range (HIGH1 through HIGH5 above and LOW1 through LOW5 below). The full sideband must be in the graticule (on screen) to be measured. Otherwise, the measurement results will be "Incomplete".

When this measurement is tracked with cursors, the cursors show the sideband being measured.

Query Syntax

```
:MEASure:FFT:ACPR? <chan_width>,<chan_spacing>,<chan>[,<source>]
```

The :MEASure:FFT:ACPR? query returns the measured Adjacent Channel Power Ratio (ACPR) value.

Return Format

```
<return_value><NL>

<return_value> ::= adjacent channel power ratio, dBV in NR3 format
```

See Also

- ":MEASure:FFT:CPOWer" on page 560
- ":MEASure:FFT:OBW" on page 561
- ":MEASure:FFT:THD" on page 562

:MEASure:FFT:CPOWer

(see page 1292)

Command Syntax

```
:MEASure:FFT:CPOWer [<source>]

<source> ::= {FUNCtion<m> | MATH<m> | FFT} (must be an FFT waveform)

<m> ::= 1 to (# math functions) in NR1 format
```

The :MEASure:FFT:CPOWer command installs an FFT analysis Channel Power measurement on screen.

Channel Power measures the spectral power across a frequency range.

The center frequency used in the measurement is the one defined for the FFT function, and the FFT span specifies the frequency range.

When this measurement is tracked with cursors, the cursors are at the left and right edges of the frequency span.

Query Syntax

```
:MEASure:FFT:CPOWer? [<source>]
```

The :MEASure:FFT:CPOWer? query returns the measured Channel Power value.

Return Format

```
<return_value><NL>

<return_value> ::= spectral channel power, dBV in NR3 format
```

See Also

- ":MEASure:FFT:ACPR" on page 559
- ":MEASure:FFT:OBW" on page 561
- ":MEASure:FFT:THD" on page 562

:MEASure:FFT:OBW

 (see page 1292)

Command Syntax

```
:MEASure:FFT:OBW <percentage>[,<source>]

<percentage> ::= percent of spectral power occupied bandwidth is
   measured for (in NR3 format)

<source> ::= {FUNCtion<m> | MATH<m> | FFT} (must be an FFT waveform)

<m> ::= 1 to (# math functions) in NR1 format
```

The :MEASure:FFT:OBW command installs an FFT analysis Occupied Bandwidth measurement on screen.

Occupied Bandwidth measures the bandwidth (frequency range) containing some percent (usually 99%) of the total spectral power. While 99% is the industry norm, you can specify the percent you want to use in the measurement.

The center frequency used in the measurement is the one defined for the FFT function, and the FFT span represents the total spectral power.

When this measurement is tracked with cursors, the cursors show the measured bandwidth (frequency range).

Query Syntax

```
:MEASure:FFT:OBW? <percentage>[,<source>]
```

The :MEASure:FFT:OBW? query returns the measured Occupied Bandwidth value.

Return Format

```
<return_value><NL>

<return_value> ::= occupied bandwidth, Hz in NR3 format
```

See Also

- ":MEASure:FFT:ACPR" on page 559
- ":MEASure:FFT:CPOWer" on page 560
- ":MEASure:FFT:THD" on page 562

:MEASure:FFT:THD

N (see page 1292)

Command Syntax

```
:MEASure:FFT:THD <tracking>[,<fundamental_freq>][,<source>]

<tracking> ::= {AUTO | MANual}

<fundamental_freq> ::= in NR3 format, required if <tracking> is MANual

<source> ::= {FUNCtion<m> | MATH<m> | FFT} (must be an FFT waveform)

<m> ::= 1 to (# math functions) in NR1 format
```

The :MEASure:FFT:THD command installs an FFT analysis Total Harmonic Distortion measurement on screen.

Total Harmonic Distortion (THD) is the ratio of power in the fundamental frequency to the power contained in the rest of the harmonics and noise. THD is a measure of signal purity.

Total Harmonic Distortion (THD) measures the power contained in the bands surrounding each harmonic and compares it to the power in the band surrounding the fundamental frequency. The width of the bands measured is the same for the fundamental frequency and each harmonic. That width is 1/2 of the fundamental frequency.

You can either enter the fundamental frequency as a measurement parameter and have the fundamental frequency and harmonics be tracked manually, or you can allow the fundamental frequency and harmonics to be tracked automatically, where the highest peak is assumed to be the fundamental frequency.

When this measurement is tracked with cursors, the cursors show the band surrounding the fundamental frequency that is being measured (at ±1/4 of the fundamental frequency).

Query Syntax

```
:MEASure:FFT:THD? <tracking>[,<fundamental_freq>][,<source>]
```

The :MEASure:FFT:THD? query returns the measured Total Harmonic Distortion value.

Return Format

```
<return_value><NL>

<return_value> ::= total harmonic distortion ratio percent in NR3 format
```

See Also

- ":MEASure:FFT:ACPR" on page 559
- ":MEASure:FFT:CPOWer" on page 560
- ":MEASure:FFT:OBW" on page 561
- "Commands Not Supported in Multiple Program Message Units" on page 1297
- "Queries Not Supported in Multiple Program Message Units" on page 1297

:MEASure:FREQuency

(see page 1292)

Command Syntax

```
:MEASure:FREQuency [<source>]

<source> ::= {<digital channels> | CHANnel<n> | FUNCtion<m> | MATH<m>
             | WMEMory<r>}

<digital channels> ::= DIGital<d>

<d> ::= 0 to (# digital channels - 1) in NR1 format

<n> ::= 1 to (# of analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

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" on page 545
- ":MEASure:SOURce" on page 599
- ":MEASure:PERiod" on page 584

Example Code

- "Example Code" on page 600

:MEASure:HISTogram:BWIDth?

(see page 1292)

Query Syntax

```
:MEASure:HISTogram:BWIDth?
```

The :MEASure:HISTogram:BWIDth? query returns the width of each bin (bucket) in the histogram.

Return Format

```
<value><NL>

<value> ::= bin width in NR3 format
```

See Also

- Chapter 21, ":HISTogram Commands," starting on page 473
- ":MEASure:HISTogram:HITS?" on page 565
- ":MEASure:HISTogram:M1S?" on page 566
- ":MEASure:HISTogram:M2S?" on page 567
- ":MEASure:HISTogram:M3S?" on page 568
- ":MEASure:HISTogram:MAXimum?" on page 569
- ":MEASure:HISTogram:MEAN?" on page 570
- ":MEASure:HISTogram:MEDian?" on page 571
- ":MEASure:HISTogram:MINimum?" on page 572
- ":MEASure:HISTogram:MODE?" on page 573
- ":MEASure:HISTogram:PEAK?" on page 574
- ":MEASure:HISTogram:PPEak?" on page 575
- ":MEASure:HISTogram:SDEViation?" on page 576

:MEASure:HISTogram:HITS?

(see page 1292)

Query Syntax `:MEASure:HISTogram:HITS?`

The :MEASure:HISTogram:HITS? query returns the sum of all hits in all bins (buckets) in the histogram.

Return Format `<value><NL>`

`<value> ::= # of total hits in NR1 format`

See Also

- Chapter 21, ":HISTogram Commands," starting on page 473
- ":MEASure:HISTogram:BWIDth?" on page 564
- ":MEASure:HISTogram:M1S?" on page 566
- ":MEASure:HISTogram:M2S?" on page 567
- ":MEASure:HISTogram:M3S?" on page 568
- ":MEASure:HISTogram:MAXimum?" on page 569
- ":MEASure:HISTogram:MEAN?" on page 570
- ":MEASure:HISTogram:MEDian?" on page 571
- ":MEASure:HISTogram:MINimum?" on page 572
- ":MEASure:HISTogram:MODE?" on page 573
- ":MEASure:HISTogram:PEAK?" on page 574
- ":MEASure:HISTogram:PPEak?" on page 575
- ":MEASure:HISTogram:SDEViation?" on page 576

:MEASure:HISTogram:M1S?

(see page 1292)

Query Syntax `:MEASure:HISTogram:M1S?`

The :MEASure:HISTogram:M1S? query returns the percentage of histogram hits that lie within ±1 Std Dev of the mean.

Return Format `<value><NL>`

`<value> ::= % of hits within +/-1 std dev in NR3 format`

See Also

- Chapter 21, ":HISTogram Commands," starting on page 473
- ":MEASure:HISTogram:BWIDth?" on page 564
- ":MEASure:HISTogram:HITS?" on page 565
- ":MEASure:HISTogram:M2S?" on page 567
- ":MEASure:HISTogram:M3S?" on page 568
- ":MEASure:HISTogram:MAXimum?" on page 569
- ":MEASure:HISTogram:MEAN?" on page 570
- ":MEASure:HISTogram:MEDian?" on page 571
- ":MEASure:HISTogram:MINimum?" on page 572
- ":MEASure:HISTogram:MODE?" on page 573
- ":MEASure:HISTogram:PEAK?" on page 574
- ":MEASure:HISTogram:PPEak?" on page 575
- ":MEASure:HISTogram:SDEViation?" on page 576

:MEASure:HISTogram:M2S?

N (see page 1292)

Query Syntax `:MEASure:HISTogram:M2S?`

The :MEASure:HISTogram:M2S? query returns the percentage of histogram hits that lie within ±2 Std Dev of the mean.

Return Format `<value><NL>`

`<value> ::= % of hits within +/-2 std devs in NR3 format`

See Also

- Chapter 21, ":HISTogram Commands," starting on page 473
- ":MEASure:HISTogram:BWIDth?" on page 564
- ":MEASure:HISTogram:HITS?" on page 565
- ":MEASure:HISTogram:M1S?" on page 566
- ":MEASure:HISTogram:M3S?" on page 568
- ":MEASure:HISTogram:MAXimum?" on page 569
- ":MEASure:HISTogram:MEAN?" on page 570
- ":MEASure:HISTogram:MEDian?" on page 571
- ":MEASure:HISTogram:MINimum?" on page 572
- ":MEASure:HISTogram:MODE?" on page 573
- ":MEASure:HISTogram:PEAK?" on page 574
- ":MEASure:HISTogram:PPEak?" on page 575
- ":MEASure:HISTogram:SDEViation?" on page 576

:MEASure:HISTogram:M3S?

(see page 1292)

Query Syntax `:MEASure:HISTogram:M3S?`

The :MEASure:HISTogram:M3S? query returns the percentage of histogram hits that lie within ±3 Std Dev of the mean.

Return Format `<value><NL>`

`<value> ::= % of hits within +/-3 std devs in NR3 format`

See Also

- Chapter 21, ":HISTogram Commands," starting on page 473
- ":MEASure:HISTogram:BWIDth?" on page 564
- ":MEASure:HISTogram:HITS?" on page 565
- ":MEASure:HISTogram:M1S?" on page 566
- ":MEASure:HISTogram:M2S?" on page 567
- ":MEASure:HISTogram:MAXimum?" on page 569
- ":MEASure:HISTogram:MEAN?" on page 570
- ":MEASure:HISTogram:MEDian?" on page 571
- ":MEASure:HISTogram:MINimum?" on page 572
- ":MEASure:HISTogram:MODE?" on page 573
- ":MEASure:HISTogram:PEAK?" on page 574
- ":MEASure:HISTogram:PPEak?" on page 575
- ":MEASure:HISTogram:SDEViation?" on page 576

:MEASure:HISTogram:MAXimum?

(see page 1292)

Query Syntax `:MEASure:HISTogram:MAXimum?`

The :MEASure:HISTogram:MAXimum? query returns the value corresponding to the maximum bin that has any hits.

Return Format `<value><NL>`

`<value> ::= value of max bin in NR3 format`

See Also

- Chapter 21, ":HISTogram Commands," starting on page 473
- ":MEASure:HISTogram:BWIDth?" on page 564
- ":MEASure:HISTogram:HITS?" on page 565
- ":MEASure:HISTogram:M1S?" on page 566
- ":MEASure:HISTogram:M2S?" on page 567
- ":MEASure:HISTogram:M3S?" on page 568
- ":MEASure:HISTogram:MEAN?" on page 570
- ":MEASure:HISTogram:MEDian?" on page 571
- ":MEASure:HISTogram:MINimum?" on page 572
- ":MEASure:HISTogram:MODE?" on page 573
- ":MEASure:HISTogram:PEAK?" on page 574
- ":MEASure:HISTogram:PPEak?" on page 575
- ":MEASure:HISTogram:SDEViation?" on page 576

:MEASure:HISTogram:MEAN?

 (see page 1292)

Query Syntax `:MEASure:HISTogram:MEAN?`

The :MEASure:HISTogram:MEAN? query returns the mean value of the histogram.

Return Format `<value><NL>`

`<value> ::= mean of histogram in NR3 format`

See Also

- Chapter 21, ":HISTogram Commands," starting on page 473
- ":MEASure:HISTogram:BWIDth?" on page 564
- ":MEASure:HISTogram:HITS?" on page 565
- ":MEASure:HISTogram:M1S?" on page 566
- ":MEASure:HISTogram:M2S?" on page 567
- ":MEASure:HISTogram:M3S?" on page 568
- ":MEASure:HISTogram:MAXimum?" on page 569
- ":MEASure:HISTogram:MEDian?" on page 571
- ":MEASure:HISTogram:MINimum?" on page 572
- ":MEASure:HISTogram:MODE?" on page 573
- ":MEASure:HISTogram:PEAK?" on page 574
- ":MEASure:HISTogram:PPEak?" on page 575
- ":MEASure:HISTogram:SDEViation?" on page 576

:MEASure:HISTogram:MEDian?

(see page 1292)

Query Syntax `:MEASure:HISTogram:MEDian?`

The :MEASure:HISTogram:MEDian? query returns the median value of the histogram.

Return Format `<value><NL>`

`<value> ::= median of histogram in NR3 format`

See Also

- Chapter 21, ":HISTogram Commands," starting on page 473
- ":MEASure:HISTogram:BWIDth?" on page 564
- ":MEASure:HISTogram:HITS?" on page 565
- ":MEASure:HISTogram:M1S?" on page 566
- ":MEASure:HISTogram:M2S?" on page 567
- ":MEASure:HISTogram:M3S?" on page 568
- ":MEASure:HISTogram:MAXimum?" on page 569
- ":MEASure:HISTogram:MEAN?" on page 570
- ":MEASure:HISTogram:MINimum?" on page 572
- ":MEASure:HISTogram:MODE?" on page 573
- ":MEASure:HISTogram:PEAK?" on page 574
- ":MEASure:HISTogram:PPEak?" on page 575
- ":MEASure:HISTogram:SDEViation?" on page 576

:MEASure:HISTogram:MINimum?

(see page 1292)

Query Syntax `:MEASure:HISTogram:MINimum?`

The :MEASure:HISTogram:MINimum? query returns the value corresponding to the minimum bin that has any hits.

Return Format `<value><NL>`

`<value> ::= value of min bin in NR3 format`

See Also

- Chapter 21, ":HISTogram Commands," starting on page 473
- ":MEASure:HISTogram:BWIDth?" on page 564
- ":MEASure:HISTogram:HITS?" on page 565
- ":MEASure:HISTogram:M1S?" on page 566
- ":MEASure:HISTogram:M2S?" on page 567
- ":MEASure:HISTogram:M3S?" on page 568
- ":MEASure:HISTogram:MAXimum?" on page 569
- ":MEASure:HISTogram:MEAN?" on page 570
- ":MEASure:HISTogram:MEDian?" on page 571
- ":MEASure:HISTogram:MODE?" on page 573
- ":MEASure:HISTogram:PEAK?" on page 574
- ":MEASure:HISTogram:PPEak?" on page 575
- ":MEASure:HISTogram:SDEViation?" on page 576

:MEASure:HISTogram:MODE?

(see page 1292)

Query Syntax `:MEASure:HISTogram:MODE?`

The :MEASure:HISTogram:MODE? query returns the mode value of the histogram.

Return Format `<value><NL>`

`<value> ::= mode of histogram in NR3 format`

See Also

- Chapter 21, ":HISTogram Commands," starting on page 473
- ":MEASure:HISTogram:BWIDth?" on page 564
- ":MEASure:HISTogram:HITS?" on page 565
- ":MEASure:HISTogram:M1S?" on page 566
- ":MEASure:HISTogram:M2S?" on page 567
- ":MEASure:HISTogram:M3S?" on page 568
- ":MEASure:HISTogram:MAXimum?" on page 569
- ":MEASure:HISTogram:MEAN?" on page 570
- ":MEASure:HISTogram:MEDian?" on page 571
- ":MEASure:HISTogram:MINimum?" on page 572
- ":MEASure:HISTogram:PEAK?" on page 574
- ":MEASure:HISTogram:PPEak?" on page 575
- ":MEASure:HISTogram:SDEViation?" on page 576

:MEASure:HISTogram:PEAK?

(see page 1292)

Query Syntax `:MEASure:HISTogram:PEAK?`

The :MEASure:HISTogram:PEAK? query returns the maximum number of hits in any single bin.

Return Format `<value><NL>`

`<value> ::= # of hits in max bin in NR1 format`

See Also

- Chapter 21, ":HISTogram Commands," starting on page 473
- ":MEASure:HISTogram:BWIDth?" on page 564
- ":MEASure:HISTogram:HITS?" on page 565
- ":MEASure:HISTogram:M1S?" on page 566
- ":MEASure:HISTogram:M2S?" on page 567
- ":MEASure:HISTogram:M3S?" on page 568
- ":MEASure:HISTogram:MAXimum?" on page 569
- ":MEASure:HISTogram:MEAN?" on page 570
- ":MEASure:HISTogram:MEDian?" on page 571
- ":MEASure:HISTogram:MINimum?" on page 572
- ":MEASure:HISTogram:MODE?" on page 573
- ":MEASure:HISTogram:PPEak?" on page 575
- ":MEASure:HISTogram:SDEViation?" on page 576

:MEASure:HISTogram:PPEak?

(see page 1292)

Query Syntax `:MEASure:HISTogram:PPEak?`

The :MEASure:HISTogram:PPEak? query returns the delta between the Max and Min values.

Return Format `<value><NL>`

`<value> ::= delta value between max bin and min bin in NR3 format`

See Also

- Chapter 21, ":HISTogram Commands," starting on page 473
- ":MEASure:HISTogram:BWIDth?" on page 564
- ":MEASure:HISTogram:HITS?" on page 565
- ":MEASure:HISTogram:M1S?" on page 566
- ":MEASure:HISTogram:M2S?" on page 567
- ":MEASure:HISTogram:M3S?" on page 568
- ":MEASure:HISTogram:MAXimum?" on page 569
- ":MEASure:HISTogram:MEAN?" on page 570
- ":MEASure:HISTogram:MEDian?" on page 571
- ":MEASure:HISTogram:MINimum?" on page 572
- ":MEASure:HISTogram:MODE?" on page 573
- ":MEASure:HISTogram:PEAK?" on page 574
- ":MEASure:HISTogram:SDEViation?" on page 576

:MEASure:HISTogram:SDEViation?

(see page 1292)

Query Syntax

```
:MEASure:HISTogram:SDEViation?
```

The :MEASure:HISTogram:SDEViation? query returns the standard deviation of the histogram values.

Return Format

```
<value><NL>

<value> ::= standard deviation of histogram in NR3 format
```

See Also

- Chapter 21, ":HISTogram Commands," starting on page 473
- ":MEASure:HISTogram:BWIDth?" on page 564
- ":MEASure:HISTogram:HITS?" on page 565
- ":MEASure:HISTogram:M1S?" on page 566
- ":MEASure:HISTogram:M2S?" on page 567
- ":MEASure:HISTogram:M3S?" on page 568
- ":MEASure:HISTogram:MAXimum?" on page 569
- ":MEASure:HISTogram:MEAN?" on page 570
- ":MEASure:HISTogram:MEDian?" on page 571
- ":MEASure:HISTogram:MINimum?" on page 572
- ":MEASure:HISTogram:MODE?" on page 573
- ":MEASure:HISTogram:PEAK?" on page 574
- ":MEASure:HISTogram:PPEak?" on page 575

:MEASure:NDUTy

N (see page 1292)

Command Syntax

```
:MEASure:NDUTy [<source>]

<source> ::= {<digital channels> | CHANnel<n> | FUNCtion<m> | MATH<m>
             | WMEMory<r>}

<digital channels> ::= DIGital<d>

<d> ::= 0 to (# digital channels - 1) in NR1 format

<n> ::= 1 to (# of analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :MEASure:NDUTy command installs a screen measurement and starts a negative 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:NDUTy? [<source>]
```

The :MEASure:NDUTy? query measures and outputs the negative duty cycle of the signal specified by the :MEASure:SOURce command. The value returned for the duty cycle is the ratio of the negative pulse width to the period. The negative 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 negative pulse width to period in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 545
- ":MEASure:PERiod" on page 584
- ":MEASure:NWIDth" on page 580
- ":MEASure:SOURce" on page 599
- ":MEASure:DUTYcycle" on page 557

:MEASure:NEDGes

(see page 1292)

Command Syntax

```
:MEASure:NEDGes [<source>]

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

<n> ::= 1 to (# analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :MEASure:NEDGes command installs a falling edge count measurement on screen. If the optional source parameter is not specified, the current source is measured.

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:NEDGes? [<source>]
```

The :MEASure:NEDGes? query measures and returns the on-screen falling edge count.

Return Format

```
<value><NL>

<value> ::= the falling edge count in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 545
- ":MEASure:SOURce" on page 599

:MEASure:NPULses

N (see page 1292)

Command Syntax

```
:MEASure:NPULses [<source>]

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>
   | <digital channels>}

<n> ::= 1 to (# analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format

<digital channels> ::= DIGital<d>

<d> ::= 0 to (# digital channels - 1) in NR1 format
```

The :MEASure:NPULses command installs a falling pulse count measurement on screen. If the optional source parameter is not specified, the current source is measured.

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:NPULses? [<source>]
```

The :MEASure:NPULses? query measures and returns the on-screen falling pulse count.

Return Format

```
<value><NL>

<value> ::= the falling pulse count in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 545
- ":MEASure:SOURce" on page 599

:MEASure:NWIDth

(see page 1292)

Command Syntax

```
:MEASure:NWIDth [<source>]

<source> ::= {<digital channels> | CHANnel<n> | FUNCtion<m> | MATH<m>
             | WMEMory<r>}

<digital channels> ::= DIGital<d>

<d> ::= 0 to (# digital channels - 1) in NR1 format

<n> ::= 1 to (# of analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :MEASure:NWIDth command installs a screen measurement and starts a negative pulse width measurement. If the optional source parameter is not 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 in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 545
- ":MEASure:SOURce" on page 599
- ":MEASure:PWIDth" on page 588
- ":MEASure:PERiod" on page 584

:MEASure:OVERshoot

(see page 1292)

Command Syntax

```
:MEASure:OVERshoot [<source>]

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

<n> ::= 1 to (# analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

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" on page 545
- ":MEASure:PREShoot" on page 587
- ":MEASure:SOURce" on page 599

- ":MEASure:VMAX" on page 621
- ":MEASure:VTOP" on page 626
- ":MEASure:VBASe" on page 620
- ":MEASure:VMIN" on page 622

:MEASure:PEDGes

(see page 1292)

Command Syntax

```
:MEASure:PEDGes [<source>]

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

<n> ::= 1 to (# analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :MEASure:PEDGes command installs a rising edge count measurement on screen. If the optional source parameter is not specified, the current source is measured.

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:PEDGes? [<source>]
```

The :MEASure:NEDGes? query measures and returns the on-screen rising edge count.

Return Format

```
<value><NL>

<value> ::= the rising edge count in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 545
- ":MEASure:SOURce" on page 599

:MEASure:PERiod

(see page 1292)

Command Syntax

```
:MEASure:PERiod [<source>]

<source> ::= {<digital channels> | CHANnel<n> | FUNCtion<m> | MATH<m>
             | WMEMory<r>}

<digital channels> ::= DIGital<d>

<d> ::= 0 to (# digital channels - 1) in NR1 format

<n> ::= 1 to (# of analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

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" on page 545
- ":MEASure:SOURce" on page 599
- ":MEASure:NWIDth" on page 580
- ":MEASure:PWIDth" on page 588
- ":MEASure:FREQuency" on page 563

Example Code

- "Example Code" on page 600

:MEASure:PHASe

N (see page 1292)

Command Syntax

```
:MEASure:PHASe [<source1>][,<source2>]

<source1>, <source2> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r
>}

<n> ::= 1 to (# analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

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" on page 545
- ":MEASure:DELay" on page 553
- ":MEASure:PERiod" on page 584
- ":MEASure:SOURce" on page 599

:MEASure:PPULses

N (see page 1292)

Command Syntax

```
:MEASure:PPULses [<source>]

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>
   | <digital channels>}

<n> ::= 1 to (# analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format

<digital channels> ::= DIGital<d>

<d> ::= 0 to (# digital channels - 1) in NR1 format
```

The :MEASure:PPULses command installs a rising pulse count measurement on screen. If the optional source parameter is not specified, the current source is measured.

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:PPULses? [<source>]
```

The :MEASure:PPULses? query measures and returns the on-screen rising pulse count.

Return Format

```
<value><NL>

<value> ::= the rising pulse count in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 545
- ":MEASure:SOURce" on page 599

:MEASure:PREShoot

(see page 1292)

Command Syntax

```
:MEASure:PREShoot [<source>]

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

<n> ::= 1 to (# analog channels) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

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" on page 545
- ":MEASure:SOURce" on page 599
- ":MEASure:VMIN" on page 622
- ":MEASure:VMAX" on page 621
- ":MEASure:VTOP" on page 626
- ":MEASure:VBASe" on page 620

:MEASure:PWIDth

(see page 1292)

Command Syntax

```
:MEASure:PWIDth [<source>]

<source> ::= {<digital channels> | CHANnel<n> | FUNCtion<m> | MATH<m>
             | WMEMory<r>}

<digital channels> ::= DIGital<d>

<d> ::= 0 to (# digital channels - 1) in NR1 format

<n> ::= 1 to (# of analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

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" on page 545
- ":MEASure:SOURce" on page 599
- ":MEASure:NWIDth" on page 580
- ":MEASure:PERiod" on page 584

:MEASure:QUICk

 (see page 1292)

Command Syntax

```
:MEASure:QUICk <source>

<source> ::= {CHANnel<n>}
```

This command installs 10 common measurements on an analog input channel source.

See Also

- "Introduction to :MEASure Commands" on page 545

:MEASure:RESults?

N (see page 1292)

Query Syntax

```
:MEASure:RESults?
```

The :MEASure:RESults? query returns the results of the continuously displayed measurements. The response to the MEASure:RESults? query is a list of comma-separated values.

If more than one measurement is running continuously, the :MEASure:RESults return values are duplicated for each continuous measurement from the first to last (top to bottom) result displayed. Each result returned is separated from the previous result by a comma. There is a maximum of 10 continuous measurements that can be continuously displayed at a time.

When no measurements are installed, the :MEASure:RESults? query returns nothing (empty string). When the count for any of the measurements is 0, the value of infinity (9.9E+37) is returned for the min, max, mean, and standard deviation.

Return Format

```
<result_list><NL>

<result_list> ::= comma-separated list of measurement results
```

The following shows the order of values received for a single measurement if :MEASure:STATistics is set to ON.

Measurement label	current	min	max	mean	std dev	count

Measurement label, current, min, max, mean, std dev, and count are only returned if :MEASure:STATistics is ON.

If :MEASure:STATistics is set to CURRent, MIN, MAX, MEAN, STDDev, or COUNt only that particular statistic value is returned for each measurement that is on.

See Also

- "Introduction to :MEASure Commands" on page 545
- ":MEASure:STATistics" on page 602
- "Queries Not Supported in Multiple Program Message Units" on page 1297

Example Code

```
' This program shows the InfiniiVision oscilloscope measurement
' results command.
' ----------------------------------------------------------------------

Option Explicit

Public myMgr As VisaComLib.ResourceManager
Public myScope As VisaComLib.FormattedIO488
Public varQueryResult As Variant
Public strQueryResult As String
```

```
#If VBA7 Then
    Private Declare PtrSafe Sub Sleep Lib "kernel32" (ByVal dwMillisecon
ds As LongPtr)
#Else
    Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Lo
ng)
#End If

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("USB0::0x2A8D::0x4704::Unset::0::INSTR")
  myScope.IO.Clear   ' Clear the interface.
  myScope.IO.Timeout = 10000   ' Set I/O communication timeout.

  ' Initialize.
  myScope.IO.Clear   ' Clear the interface.
  myScope.WriteString "*RST"   ' Reset to the defaults.
  myScope.WriteString "*CLS"   ' Clear the status data structures.
  myScope.WriteString ":AUToscale"

  ' Install some measurements.
  myScope.WriteString ":MEASure:SOURce CHANnel2"   ' Input source.

  Dim MeasurementArray(3) As String
  MeasurementArray(0) = "FREQuency"
  MeasurementArray(1) = "DUTYcycle"
  MeasurementArray(2) = "VAMPlitude"
  MeasurementArray(3) = "VPP"
  Dim Measurement As Variant

  For Each Measurement In MeasurementArray
    myScope.WriteString ":MEASure:" + Measurement
    myScope.WriteString ":MEASure:" + Measurement + "?"
    strQueryResult = myScope.ReadString   ' Read measurement value.
    Debug.Print Measurement + ": " + strQueryResult
  Next

  myScope.WriteString ":MEASure:STATistics:RESet"   ' Reset stats.
  Sleep 5000   ' Wait for 5 seconds.

  ' Select the statistics results type.
  Dim ResultsTypeArray(6) As String
  ResultsTypeArray(0) = "CURRent"
  ResultsTypeArray(1) = "MINimum"
  ResultsTypeArray(2) = "MAXimum"
  ResultsTypeArray(3) = "MEAN"
  ResultsTypeArray(4) = "STDDev"
  ResultsTypeArray(5) = "COUNt"
  ResultsTypeArray(6) = "ON"   ' All results.
  Dim ResultType As Variant
```

```
Dim ResultsList()

Dim ValueColumnArray(6) As String
ValueColumnArray(0) = "Meas_Lbl"
ValueColumnArray(1) = "Current"
ValueColumnArray(2) = "Min"
ValueColumnArray(3) = "Max"
ValueColumnArray(4) = "Mean"
ValueColumnArray(5) = "Std_Dev"
ValueColumnArray(6) = "Count"
Dim ValueColumn As Variant

For Each ResultType In ResultsTypeArray
  myScope.WriteString ":MEASure:STATistics " + ResultType

  ' Get the statistics results.
  Dim intCounter As Integer
  intCounter = 0
  myScope.WriteString ":MEASure:RESults?"
  ResultsList() = myScope.ReadList

  For Each Measurement In MeasurementArray

    If ResultType = "ON" Then   ' All statistics.

      For Each ValueColumn In ValueColumnArray
        If VarType(ResultsList(intCounter)) <> vbString Then
          Debug.Print "Measure statistics result CH1, " + _
              Measurement + ", "; ValueColumn + ": " + _
              FormatNumber(ResultsList(intCounter), 4)

        Else   ' Result is a string (e.g., measurement label).
          Debug.Print "Measure statistics result CH1, " + _
              Measurement + ", "; ValueColumn + ": " + _
              ResultsList(intCounter)

        End If

        intCounter = intCounter + 1

      Next

    Else   ' Specific statistic (e.g., Current, Max, Min, etc.).

      Debug.Print "Measure statistics result CH1, " + _
          Measurement + ", "; ResultType + ": " + _
          FormatNumber(ResultsList(intCounter), 4)

      intCounter = intCounter + 1

    End If

  Next

Next

Exit Sub
```

```
VisaComError:
  MsgBox "VISA COM Error:" + vbCrLf + Err.Description

End Sub
```

:MEASure:RISetime

 (see page 1292)

Command Syntax

```
:MEASure:RISetime [<source>]

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

<n> ::= 1 to (# analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

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 timebase 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" on page 545
- ":MEASure:SOURce" on page 599
- ":MEASure:FALLtime" on page 558
- ":MEASure:SLEWrate" on page 597

:MEASure:SDEViation

N (see page 1292)

Command Syntax

```
:MEASure:SDEViation [<source>]
<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}
<n> ::= 1-2 or 1-4 (# of analog channels) in NR1 format
<m> ::= 1 to (# math functions) in NR1 format
<r> ::= 1 to (# ref waveforms) in NR1 format
```

NOTE This ":MEASure:VRMS DISPlay, AC" command is the preferred syntax for making standard deviation measurements.

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" on page 545
- ":MEASure:VRMS" on page 625
- ":MEASure:SOURce" on page 599

:MEASure:SHOW

 (see page 1292)

Command Syntax

```
:MEASure:SHOW <on_off>

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

The :MEASure:SHOW command enables markers for tracking measurements on the display.

Query Syntax

```
:MEASure:SHOW?
```

The :MEASure:SHOW? query returns the current state of the markers.

This can return OFF when :MARKer:MODE selects a mode other than MEASurement.

Return Format

```
<on_off><NL>

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

See Also

- "Introduction to :MEASure Commands" on page 545
- ":MARKer:MODE" on page 511

:MEASure:SLEWrate

N (see page 1292)

Command Syntax

```
:MEASure:SLEWrate [<source>[,<slope>]]

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

<n> ::= 1 to (# of analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format

<slope> ::= {RISing | FALLing | EITHer}
```

The :MEASure:SLEWrate command installs a slew rate measurement on the oscilloscope's front-panel screen.

The slew rate of an edge is the change in vertical value between the lower and upper thresholds of a waveform source (typically volts) divided by the change in time between where the waveform threshold crossings take place.

The lower and upper thresholds for the measurement are specified by the measurement threshold settings for the source waveform (see :MEASure:DEFine). If the <source> parameter is not specified, the current :MEASure:SOURce setting is used

The on-screen edge closest to the timebase reference is measured.

You can specify that a RISing edge be measured, a FALLing edge be measured, or either a rising or a falling edge be measured (EITHer). If the <slope> is not specified, a rising edge is measured.

NOTE This command is not available if the source is a FFT (Fast Fourier Transform) waveform.

Query Syntax

```
:MEASure:SLEWrate? [<source>[,<slope>]]
```

The :MEASure:SLEWrate? query returns the measured slew rate of the specified edge closest to the timebase reference.

Return Format

```
<value><NL>

<value> ::= slew rate in vertical units/second in NR3 format
```

See Also

- ":MEASure:DELay:DEFine" on page 555
- ":MEASure:THResholds:ABSolute" on page 612
- ":MEASure:THResholds:METHod" on page 613
- ":MEASure:THResholds:PERCent" on page 614
- ":MEASure:THResholds:STANdard" on page 615

- ":MEASure:SOURce" on page 599
- ":MEASure:RISetime" on page 594
- ":MEASure:FALLtime" on page 558
- "Introduction to :MEASure Commands" on page 545

:MEASure:SOURce

(see page 1292)

Command Syntax

```
:MEASure:SOURce <source1>[,<source2>]

<source1>,<source2> ::= {<digital channels> | CHANnel<n> | FUNCtion<m>
   | MATH<m> | WMEMory<r>}

<digital channels> ::= DIGital<d>

<n> ::= 1 to (# of analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1-2 in NR1 format

<d> ::= 0 to (# digital channels - 1) in NR1 format
```

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<m>, or MATH<m> 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 applies only to :MEASure:DELay and :MEASure:PHASe measurements.

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

NOTE FUNCtion or MATH (without the "<m>" math function number) is an alias for FUNCtion2 or MATH2. The query will return FUNC2 if the source is FUNCtion or MATH.

Return Format

```
<source1>,<source2><NL>

<source1>,<source2> ::= {<digital channels> | CHAN<n> | FUNC<m>
   | WMEM<r> | NONE}
```

See Also:

- "Introduction to :MEASure Commands" on page 545
- ":MARKer:MODE" on page 511

- ":MARKer:X1Y1source" on page 514
- ":MARKer:X2Y2source" on page 517
- ":MEASure:DELay" on page 553
- ":MEASure:PHASe" on page 585

Example Code

```
' This program shows InfiniiVision oscilloscope measure commands.
' ----------------------------------------------------------------------

Option Explicit

Public myMgr As VisaComLib.ResourceManager
Public myScope As VisaComLib.FormattedIO488
Public varQueryResult As Variant
Public strQueryResult As String

#If VBA7 Then
    Private Declare PtrSafe Sub Sleep Lib "kernel32" (ByVal dwMillisecon
ds As LongPtr)
#Else
    Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Lo
ng)
#End If

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("USB0::0x2A8D::0x4704::Unset::0::INSTR")
  myScope.IO.Clear   ' Clear the interface.
  myScope.IO.Timeout = 10000   ' Set I/O communication timeout.

  ' Initialize.
  myScope.IO.Clear   ' Clear the interface.
  myScope.WriteString "*RST"   ' Reset to the defaults.
  myScope.WriteString "*CLS"   ' Clear the status data structures.
  myScope.WriteString ":AUToscale"

  ' 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 Pk to Pk 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"

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error:" + vbCrLf + Err.Description

End Sub
```

See complete example programs at: Chapter 44, “Programming Examples,” starting on page 1301

:MEASure:STATistics

(see page 1292)

Command Syntax

```
:MEASure:STATistics <type>

<type> ::= {{ON | 1} | CURRent | MINimum | MAXimum | MEAN | STDDev
           | COUNt}
```

The :MEASure:STATistics command determines the type of information returned by the :MEASure:RESults? query. ON means all the statistics are on.

Query Syntax

```
:MEASure:STATistics?
```

The :MEASure:STATistics? query returns the current statistics mode.

Return Format

```
<type><NL>

<type> ::= {ON | CURR | MIN | MAX | MEAN | STDD | COUN}
```

See Also

- "Introduction to :MEASure Commands" on page 545
- ":MEASure:RESults?" on page 590
- ":MEASure:STATistics:DISPlay" on page 603
- ":MEASure:STATistics:RESet" on page 606
- ":MEASure:STATistics:INCRement" on page 604

Example Code

- "Example Code" on page 590

:MEASure:STATistics:DISPlay

N (see page 1292)

Command Syntax `:MEASure:STATistics:DISPlay {{0 | OFF} | {1 | ON}}`

The :MEASure:STATistics:DISPlay command disables or enables the display of the measurement statistics.

Query Syntax `:MEASure:STATistics:DISPlay?`

The :MEASure:STATistics:DISPlay? query returns the state of the measurement statistics display.

Return Format `{0 | 1}<NL>`

See Also

- "Introduction to :MEASure Commands" on page 545
- ":MEASure:RESults?" on page 590
- ":MEASure:STATistics" on page 602
- ":MEASure:STATistics:MCOunt" on page 605
- ":MEASure:STATistics:RESet" on page 606
- ":MEASure:STATistics:INCRement" on page 604
- ":MEASure:STATistics:RSDeviation" on page 607

:MEASure:STATistics:INCRement

 (see page 1292)

Command Syntax `:MEASure:STATistics:INCRement`

This command updates the statistics once (incrementing the count by one) using the current measurement values.This command lets you, for example, gather statistics over multiple pulses captured in a single acquisition. To do this, change the horizontal position and enter the command for each new pulse that is measured.

This command is allowed only when the oscilloscope is stopped and measurements are on.

The command is allowed in segmented acquisition mode.

See Also

- "Introduction to :MEASure Commands" on page 545
- ":MEASure:STATistics" on page 602
- ":MEASure:STATistics:DISPlay" on page 603
- ":MEASure:STATistics:RESet" on page 606
- ":MEASure:RESults?" on page 590

:MEASure:STATistics:MCOunt

(see page 1292)

Command Syntax

```
:MEASure:STATistics:MCOunt <setting>

<setting> ::= {INFinite | <count>}

<count> ::= 2 to 2000 in NR1 format
```

The :MEASure:STATistics:MCOunt command specifies the maximum number of values used when calculating measurement statistics.

Query Syntax

```
:MEASure:STATistics:MCOunt?
```

The :MEASure:STATistics:MCOunt? query returns the current measurement statistics max count setting.

Return Format

```
<setting><NL>

<setting> ::= {INF | <count>}

<count> ::= 2 to 2000
```

See Also

- "Introduction to :MEASure Commands" on page 545
- ":MEASure:RESults?" on page 590
- ":MEASure:STATistics" on page 602
- ":MEASure:STATistics:DISPlay" on page 603
- ":MEASure:STATistics:RSDeviation" on page 607
- ":MEASure:STATistics:RESet" on page 606
- ":MEASure:STATistics:INCRement" on page 604

:MEASure:STATistics:RESet

 (see page 1292)

Command Syntax `:MEASure:STATistics:RESet`

This command resets the measurement statistics, zeroing the counts.

Note that the measurement (statistics) configuration is not deleted.

See Also

- "Introduction to :MEASure Commands" on page 545
- ":MEASure:STATistics" on page 602
- ":MEASure:STATistics:DISPlay" on page 603
- ":MEASure:RESults?" on page 590
- ":MEASure:STATistics:INCRement" on page 604

Example Code

- "Example Code" on page 590

:MEASure:STATistics:RSDeviation

(see page 1292)

Command Syntax `:MEASure:STATistics:RSDeviation {{0 | OFF} | {1 | ON}}`

The :MEASure:STATistics:RSDeviation command disables or enables relative standard deviations, that is, standard deviation/mean, in the measurement statistics.

Query Syntax `:MEASure:STATistics:RSDeviation?`

The :MEASure:STATistics:RSDeviation? query returns the current relative standard deviation setting.

Return Format `{0 | 1}<NL>`

See Also

- "Introduction to :MEASure Commands" on page 545
- ":MEASure:RESults?" on page 590
- ":MEASure:STATistics" on page 602
- ":MEASure:STATistics:DISPlay" on page 603
- ":MEASure:STATistics:MCOunt" on page 605
- ":MEASure:STATistics:RESet" on page 606
- ":MEASure:STATistics:INCRement" on page 604

:MEASure:TEDGe

N (see page 1292)

Command Syntax

```
:MEASure:TEDGe [<slope>,]<occurrence>[,<source>]

<slope> ::= {RISing | FALLing | BOTH}

<occurrence> ::= [+ | -]<number>

<number> ::= the edge number in NR1 format

<source> ::= {<digital channels> | CHANnel<n> | FUNCtion<m> | MATH<m>
   | WMEMory<r>}

<digital channels> ::= DIGital<d>

<n> ::= 1 to (# of analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format

<d> ::= 0 to (# digital channels - 1) in NR1 format
```

The :MEASure:TEDGe command installs a screen measurement whose result is the time of the specified edge transition relative to the trigger edge (time=zero).

The threshold voltage used for this measurement is at the 50% point with a small amount of hysteresis added. You cannot change the threshold voltage used for this measurement with the :MEASure:DEFine command.

The optional <slope> parameter specifies whether to look for RISing edges or FALLing edges. If not otherwise specified, rising edges are assumed.

The <occurrence> parameter specifies the Nth edge to identify.

An <occurrence> value of zero (0) means the edge closest to the timebase reference; it is equivalent to the **Auto** selection in in the graphical user interface. The BOTH (both edges) <slope> option can be used only when the <occurrence> count is zero (0).

Any other <occurrence> value means the Nth edge from the left side of the display.

The <occurrence> parameter can optionally have a plus sign (+) or a minus sign (-) to specify a rising or falling slope. Plus or minus signs cannot be used when the <slope> parameter is used.

If the optional <source> parameter is not specified, the current :MEASure:SOURce setting is used.

NOTE This command is not available if the source is a FFT (Fast Fourier Transform) waveform.

Query Syntax

```
:MEASure:TEDGe? [<slope>,]<occurrence>[,<source>]
```

When the :MEASure:TEDGe query is sent, the displayed signal is searched for the specified transition.

The time interval between the trigger event and the edge occurrence is returned.

If the specified crossing cannot be found, the oscilloscope returns +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.

Return Format

```
<value><NL>

<value> ::= time in seconds of the specified transition in NR3 format
```

:MEASure:TEDGe Code

You can use multiple :MEASure:TEDGe? measurements to form delay and phase measurements:

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 example:

```
' This program shows the InfiniiVision oscilloscope :MEASure:TEDGe
' command.
' ----------------------------------------------------------------------

Option Explicit

Public myMgr As VisaComLib.ResourceManager
Public myScope As VisaComLib.FormattedIO488
Public varQueryResult As Variant
Public strQueryResult As String

#If VBA7 Then
    Private Declare PtrSafe Sub Sleep Lib "kernel32" (ByVal dwMillisecon
ds As LongPtr)
#Else
    Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Lo
ng)
#End If

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("USB0::0x2A8D::0x4704::Unset::0::INSTR")
  myScope.IO.Clear   ' Clear the interface.
  myScope.IO.Timeout = 10000   ' Set I/O communication timeout.

  ' Initialize.
```

```
  myScope.IO.Clear   ' Clear the interface.
  myScope.WriteString "*RST"   ' Reset to the defaults.
  myScope.WriteString "*CLS"   ' Clear the status data structures.
  myScope.WriteString ":AUToscale"

  ' 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

  ' Query time at 1st rising edge on ch1.
  myScope.WriteString ":MEASure:TEDGe? +1, CHAN1"

  ' Read time at edge 1 on ch 1.
  dblChan1Edge1 = myScope.ReadNumber

  ' Query time at 1st rising edge on ch2.
  myScope.WriteString ":MEASure:TEDGe? +1, CHAN2"

  ' Read time at edge 1 on ch 2.
  dblChan2Edge1 = myScope.ReadNumber

  ' Calculate delay time between ch1 and ch2.
  dblDelay = dblChan2Edge1 - dblChan1Edge1

  ' Write calculated delay time to screen.
  MsgBox "Delay = " + vbCrLf + CStr(dblDelay)

  ' Make a phase difference measurement between channel 1 and 2.
  ' Query time at 1st rising edge on ch1.
  myScope.WriteString ":MEASure:TEDGe? +2, CHAN1"

  ' Read time at edge 2 on ch 1.
  dblChan1Edge2 = myScope.ReadNumber

  ' Calculate period of ch 1.
  dblPeriod = dblChan1Edge2 - dblChan1Edge1

  ' Calculate phase difference between ch1 and ch2.
  dblPhase = (dblDelay / dblPeriod) * 360
  MsgBox "Phase = " + vbCrLf + CStr(dblPhase)

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error:" + vbCrLf + Err.Description

End Sub
```

See complete example programs at: Chapter 44, “Programming Examples,” starting on page 1301

See Also

- "Introduction to :MEASure Commands" on page 545

- ":MEASure:TVALue?" on page 616
- ":MEASure:YATX" on page 630
- "Commands Not Supported in Multiple Program Message Units" on page 1297
- "Queries Not Supported in Multiple Program Message Units" on page 1297

:MEASure:THResholds:ABSolute

N (see page 1292)

Command Syntax

```
:MEASure:THResholds:ABSolute <upper>,<middle>,<lower>[,<source>]

<upper>,<middle>,<lower> ::= A number specifying the upper, middle,
   and lower threshold absolute values in NR3 format.

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

<n> ::= 1 to (# analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :MEASure:THResholds:ABSolute command sets the upper, middle, and lower 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

When <source> is not included in the command or query, the source identified by :MEASure:SOURce is used.

The ":MEASure:DEFine THResholds" command also lets you specify absolute measurement thresholds.

Query Syntax

```
:MEASure:THResholds:ABSolute? [<source>]
```

The :MEASure:THResholds:ABSolute? query returns the upper, middle, and lower measurement threshold absolute values.

Return Format

```
<upper>,<middle>,<lower><NL>

<upper>,<middle>,<lower> ::= A number specifying the upper, middle,
   and lower threshold absolute values in NR3 format.
```

See Also

- ":MEASure:THResholds:METHod" on page 613
- ":MEASure:THResholds:PERCent" on page 614
- ":MEASure:THResholds:STANdard" on page 615
- ":CHANnel<n>:RANGe" on page 295
- ":CHANnel<n>:SCALe" on page 296
- ":CHANnel<n>:PROBe" on page 280
- ":CHANnel<n>:UNITs" on page 297
- ":MEASure:SOURce" on page 599
- ":MEASure:DELay:DEFine" on page 555

:MEASure:THResholds:METHod

N (see page 1292)

Command Syntax

```
:MEASure:THResholds:METHod <threshold_mode>[,<source>]

<threshold_mode> ::= {PERCent | ABSolute}

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

<n> ::= 1 to (# analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :MEASure:THResholds:METHod command specifies the method for entering measurement threshold values:

- PERCent — Lets you set the measurement thresholds to user-defined percentages between 5% and 95% of values between Vbase and Vtop. See :MEASure:THResholds:PERCent.
- ABSolute — Lets you set 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. See :MEASure:THResholds:ABSolute.

When <source> is not included in the command or query, the source identified by :MEASure:SOURce is used.

The ":MEASure:DEFine THResholds" command also lets you specify percent or absolute measurement thresholds.

Query Syntax

```
:MEASure:THResholds:METHod? [<source>]
```

The :MEASure:THResholds:METHod? query returns the selected measurement threshold entry method.

Return Format

```
<threshold_mode><NL>

<threshold_mode> ::= {PERC | ABS}
```

See Also

- ":MEASure:THResholds:ABSolute" on page 612
- ":MEASure:THResholds:PERCent" on page 614
- ":MEASure:THResholds:STANdard" on page 615
- ":MEASure:SOURce" on page 599
- ":MEASure:DELay:DEFine" on page 555

:MEASure:THResholds:PERCent

(see page 1292)

Command Syntax

```
:MEASure:THResholds:PERCent <upper>,<middle>,<lower>[,<source>]

<upper>,<middle>,<lower> ::= A number specifying the upper, middle,
   and lower threshold percentage values between Vbase and Vtop
   in NR3 format.

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

<n> ::= 1 to (# analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :MEASure:THResholds:PERCent command sets the upper, middle, and lower measurement thresholds to user-defined percentages between 5% and 95% of values between Vbase and Vtop.

When <source> is not included in the command or query, the source identified by :MEASure:SOURce is used.

The ":MEASure:DEFine THResholds" command also lets you specify percent measurement thresholds.

Query Syntax

```
:MEASure:THResholds:PERCent? [<source>]
```

The :MEASure:THResholds:PERCent? query returns the upper, middle, and lower measurement threshold percent values.

Return Format

```
<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.
```

See Also

- ":MEASure:THResholds:ABSolute" on page 612
- ":MEASure:THResholds:METHod" on page 613
- ":MEASure:THResholds:STANdard" on page 615
- ":MEASure:SOURce" on page 599
- ":MEASure:DELay:DEFine" on page 555

:MEASure:THResholds:STANdard

(see page 1292)

Command Syntax

```
:MEASure:THResholds:STANdard [<source>]

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

<n> ::= 1 to (# analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :MEASure:THResholds:STANdard command sets the lower, middle, and upper measurement thresholds to 10%, 50%, and 90% values between Vbase and Vtop.

When <source> is not included in the command, the source identified by :MEASure:SOURce is used.

The ":MEASure:DEFine THResholds" command also lets you set standard measurement thresholds.

See Also

- ":MEASure:THResholds:ABSolute" on page 612
- ":MEASure:THResholds:METHod" on page 613
- ":MEASure:THResholds:PERCent" on page 614
- ":MEASure:SOURce" on page 599
- ":MEASure:DELay:DEFine" on page 555

:MEASure:TVALue?

 (see page 1292)

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<m> | MATH<m> | WMEMory<r>}

<n> ::= 1 to (# analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

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" on page 545
- ":MEASure:TEDGe" on page 608
- ":MEASure:YATX" on page 630

:MEASure:VAMPlitude

(see page 1292)

Command Syntax

```
:MEASure:VAMPlitude [<source>]

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

<n> ::= 1 to (# analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

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" on page 545
- ":MEASure:SOURce" on page 599
- ":MEASure:VBASe" on page 620
- ":MEASure:VTOP" on page 626
- ":MEASure:VPP" on page 623

:MEASure:VAVerage

(see page 1292)

Command Syntax

```
:MEASure:VAVerage [<interval>][,<source>]

<interval> ::= {CYCLe | DISPlay}

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | FFT | WMEMory<r>}

<n> ::= 1-2 or 1-4 (# of analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :MEASure:VAVerage command installs a screen measurement and starts an average value measurement.

The <interval> option lets you specify the measurement interval: either an integral number of cycles, or the full screen. If <interval> is not specified, DISPlay is implied.

If the optional source parameter is specified, the current source is modified.

Query Syntax

```
:MEASure:VAVerage? [<interval>][,<source>]
```

The :MEASure:VAVerage? query returns the average value of an integral number of periods of the signal.

Return Format

```
<value><NL>

<value> ::= calculated average value in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 545
- ":MEASure:SOURce" on page 599

:MEASure:VBASe

(see page 1292)

Command Syntax

```
:MEASure:VBASe [<source>]

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

<n> ::= 1 to (# analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

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" on page 545
- ":MEASure:SOURce" on page 599
- ":MEASure:VTOP" on page 626
- ":MEASure:VAMPlitude" on page 618
- ":MEASure:VMIN" on page 622

:MEASure:VMAX

(see page 1292)

Command Syntax

```
:MEASure:VMAX [<source>]

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | FFT | WMEMory<r>}

<n> ::= 1-2 or 1-4 (# of analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

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" on page 545
- ":MEASure:SOURce" on page 599
- ":MEASure:VMIN" on page 622
- ":MEASure:VPP" on page 623
- ":MEASure:VTOP" on page 626

:MEASure:VMIN

(see page 1292)

Command Syntax

```
:MEASure:VMIN [<source>]

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | FFT | WMEMory<r>}

<n> ::= 1 to (# analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

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" on page 545
- ":MEASure:SOURce" on page 599
- ":MEASure:VBASe" on page 620
- ":MEASure:VMAX" on page 621
- ":MEASure:VPP" on page 623

:MEASure:VPP

(see page 1292)

Command Syntax

```
:MEASure:VPP [<source>]

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | FFT | WMEMory<r>}

<n> ::= 1 to (# analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

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" on page 545
- ":MEASure:SOURce" on page 599
- ":MEASure:VMAX" on page 621
- ":MEASure:VMIN" on page 622
- ":MEASure:VAMPlitude" on page 618

:MEASure:VRATio

(see page 1292)

Command Syntax

```
:MEASure:VRATio [<interval>][,<source1>][,<source2>]

<interval> ::= {CYCLe | DISPlay}

<source1,2> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

<n> ::= 1 to (# analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :MEASure:VRATio command installs a ratio measurement on screen. Ratio measurements show the ratio of the ACRMS value of source1 to that of source2, expressed in dB.

The <interval> option lets you specify the measurement interval: either an integral number of cycles, or the full screen. If <interval> is not specified, DISPlay is implied.

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:VRATio? [<interval>][<source1>][,<source2>]
```

The :MEASure:VRATio? query measures and returns the ratio of AC RMS values of the specified sources expressed as dB.

Return Format

```
<value><NL>

<value> ::= the ratio value in dB in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 545
- ":MEASure:VRMS" on page 625
- ":MEASure:SOURce" on page 599

:MEASure:VRMS

(see page 1292)

Command Syntax

```
:MEASure:VRMS [<interval>][,<type>][,<source>]

<interval> ::= {CYCLe | DISPlay}

<type> ::= {AC | DC}

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

<n> ::= 1-2 or 1-4 (# of analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :MEASure:VRMS command installs a screen measurement and starts an RMS value measurement.

The <interval> option lets you specify the measurement interval: either an integral number of cycles, or the full screen. If <interval> is not specified, DISPlay is implied.

The <type> option lets you choose between a DC RMS measurement and an AC RMS measurement. If <type> is not specified, DC is implied.

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? [<interval>][,<type>][,<source>]
```

The :MEASure:VRMS? query measures and outputs the RMS measurement value.

Return Format

```
<value><NL>

<value> ::= calculated dc RMS value in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 545
- ":MEASure:SOURce" on page 599

:MEASure:VTOP

 (see page 1292)

Command Syntax

```
:MEASure:VTOP [<source>]

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

<n> ::= 1-2 or 1-4 (# of analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

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" on page 545
- ":MEASure:SOURce" on page 599
- ":MEASure:VMAX" on page 621
- ":MEASure:VAMPlitude" on page 618
- ":MEASure:VBASe" on page 620

:MEASure:WINDow

N (see page 1292)

Command Syntax

```
:MEASure:WINDow <type>

<type> ::= {MAIN | ZOOM | AUTO | GATE}
```

The :MEASure:WINDow command lets you choose whether measurements are made in the Main window portion of the display, the Zoom window portion of the display (when the zoomed time base is displayed), or gated by the X1 and X2 cursors.

- MAIN — the measurement window is the Main window.
- ZOOM — the measurement window is the lower, Zoom window.
- AUTO — when the zoomed time base is displayed, the measurement is attempted in the lower, Zoom window; if it cannot be made there, or if the zoomed time base is not displayed, the Main window is used.
- GATE — the measurement window is between the X1 and X2 cursors. When the zoomed time base is displayed, the X1 and X2 cursors in the Zoom window portion of the display are used.

Query Syntax

```
:MEASure:WINDow?
```

The :MEASure:WINDow? query returns the current measurement window setting.

Return Format

```
<type><NL>

<type> ::= {MAIN | ZOOM | AUTO | GATE}
```

See Also

- "Introduction to :MEASure Commands" on page 545
- ":MEASure:SOURce" on page 599

:MEASure:XMAX

(see page 1292)

Command Syntax

```
:MEASure:XMAX [<source>]

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | FFT | WMEMory<r>}

<n> ::= 1-2 or 1-4 (# of analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

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.

Return Format

```
<value><NL>

<value> ::= horizontal value of the maximum in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 545
- ":MEASure:XMIN" on page 629

:MEASure:XMIN

N (see page 1292)

Command Syntax

```
:MEASure:XMIN [<source>]

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | FFT | WMEMory<r>}

<n> ::= 1-2 or 1-4 (# of analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

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.

Return Format

```
<value><NL>

<value> ::= horizontal value of the minimum in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 545
- ":MEASure:XMAX" on page 628

:MEASure:YATX

N (see page 1292)

Command Syntax

```
:MEASure:YATX <horiz_location>[,<source>]

<horiz_location> ::= time from trigger in seconds

<source> ::= {<digital channels> | CHANnel<n> | FUNCtion<m> | MATH<m>
             | FFT | WMEMory<r>}

<digital channels> ::= DIGital<d>

<d> ::= 0 to (# digital channels - 1) in NR1 format

<n> ::= 1 to (# of analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :MEASure:YATX command installs a screen measurement of the vertical value at a specified horizontal value on the source specified (see also :MEASure:SOURce).

The specified horizontal value must be on the screen; when it is a time value, it is referenced to the trigger event. If the optional source parameter is specified, the measurement source is modified.

NOTE When the source is an FFT (Fast Fourier Transform) waveform, the <horiz_location> is a frequency value instead of a time value.

Query Syntax

```
:MEASure:YATX? <horiz_location>[,<source>]
```

The :MEASure:YATX? query returns the vertical value at a specified horizontal value on the source specified (see also :MEASure:SOURce).

Return Format

```
<value><NL>

<value> ::= vertical value at the specified horizontal location
            in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 545
- ":MEASure:SOURce" on page 599
- ":MEASure:TEDGe" on page 608
- ":MEASure:TVALue?" on page 616

26 :MTESt Commands

The MTESt subsystem commands and queries control the mask test features. See "Introduction to :MTESt Commands" on page 633.

Table 91 :MTESt Commands Summary

Command	Query	Options and Query Returns
:MTESt:ALL {{0 | OFF} | {1 | ON}} (see page 637)	:MTESt:ALL? (see page 637)	{0 | 1}
:MTESt:AMASk:CREate (see page 638)	n/a	n/a
:MTESt:AMASk:SOURce <source> (see page 639)	:MTESt:AMASk:SOURce? (see page 639)	<source> ::= CHANnel<n> <n> ::= {1 | 2 | 3 | 4} for 4ch models <n> ::= {1 | 2} for 2ch models
:MTESt:AMASk:UNITs <units> (see page 640)	:MTESt:AMASk:UNITs? (see page 640)	<units> ::= {CURRent | DIVisions}
:MTESt:AMASk:XDELta <value> (see page 641)	:MTESt:AMASk:XDELta? (see page 641)	<value> ::= X delta value in NR3 format
:MTESt:AMASk:YDELta <value> (see page 642)	:MTESt:AMASk:YDELta? (see page 642)	<value> ::= Y delta value in NR3 format
n/a	:MTESt:COUNt:FWAVeforms? [CHANnel<n>] (see page 643)	<failed> ::= number of failed waveforms in NR1 format
:MTESt:COUNt:RESet (see page 644)	n/a	n/a
n/a	:MTESt:COUNt:TIME? (see page 645)	<time> ::= elapsed seconds in NR3 format
n/a	:MTESt:COUNt:WAVeforms? (see page 646)	<count> ::= number of waveforms in NR1 format
:MTESt:DATA <mask> (see page 647)	:MTESt:DATA? (see page 647)	<mask> ::= data in IEEE 488.2 # format.

Table 91 :MTESt Commands Summary (continued)

Command	Query	Options and Query Returns
:MTESt:DELete (see page 648)	n/a	n/a
:MTESt:ENABle {{0 | OFF} | {1 | ON}} (see page 649)	:MTESt:ENABle? (see page 649)	{0 | 1}
:MTESt:LOCK {{0 | OFF} | {1 | ON}} (see page 650)	:MTESt:LOCK? (see page 650)	{0 | 1}
:MTESt:RMODe <rmode> (see page 651)	:MTESt:RMODe? (see page 651)	<rmode> ::= {FOREver | TIME | SIGMa | WAVeforms}
:MTESt:RMODe:FACTion:MEASure {{0 | OFF} | {1 | ON}} (see page 652)	:MTESt:RMODe:FACTion:MEASure? (see page 652)	{0 | 1}
:MTESt:RMODe:FACTion:PRINt {{0 | OFF} | {1 | ON}} (see page 653)	:MTESt:RMODe:FACTion:PRINt? (see page 653)	{0 | 1}
:MTESt:RMODe:FACTion:SAVE {{0 | OFF} | {1 | ON}} (see page 654)	:MTESt:RMODe:FACTion:SAVE? (see page 654)	{0 | 1}
:MTESt:RMODe:FACTion:STOP {{0 | OFF} | {1 | ON}} (see page 655)	:MTESt:RMODe:FACTion:STOP? (see page 655)	{0 | 1}
:MTESt:RMODe:SIGMa <level> (see page 656)	:MTESt:RMODe:SIGMa? (see page 656)	<level> ::= from 0.1 to 9.3 in NR3 format
:MTESt:RMODe:TIME <seconds> (see page 657)	:MTESt:RMODe:TIME? (see page 657)	<seconds> ::= from 1 to 86400 in NR3 format
:MTESt:RMODe:WAVeforms <count> (see page 658)	:MTESt:RMODe:WAVeforms? (see page 658)	<count> ::= number of waveforms in NR1 format
:MTESt:SCALe:BIND {{0 | OFF} | {1 | ON}} (see page 659)	:MTESt:SCALe:BIND? (see page 659)	{0 | 1}
:MTESt:SCALe:X1 <x1_value> (see page 660)	:MTESt:SCALe:X1? (see page 660)	<x1_value> ::= X1 value in NR3 format
:MTESt:SCALe:XDELta <xdelta_value> (see page 661)	:MTESt:SCALe:XDELta? (see page 661)	<xdelta_value> ::= X delta value in NR3 format

Table 91 :MTESt Commands Summary (continued)

Command	Query	Options and Query Returns
:MTESt:SCALe:Y1 <y1_value> (see page 662)	:MTESt:SCALe:Y1? (see page 662)	<y1_value> ::= Y1 value in NR3 format
:MTESt:SCALe:Y2 <y2_value> (see page 663)	:MTESt:SCALe:Y2? (see page 663)	<y2_value> ::= Y2 value in NR3 format
:MTESt:SOURce <source> (see page 664)	:MTESt:SOURce? (see page 664)	<source> ::= {CHANnel<n> | NONE} <n> ::= {1 | 2 | 3 | 4} for 4ch models <n> ::= {1 | 2} for 2ch models
n/a	:MTESt:TITLe? (see page 665)	<title> ::= a quoted string of up to 128 ASCII characters

Introduction to :MTESt Commands

Mask testing automatically compares the current displayed waveform with the boundaries of a set of polygons that you define. Any waveform or sample that falls within the boundaries of one or more polygons is recorded as a failure.

Reporting the Setup

Use :MTESt? to query setup information for the MTESt subsystem.

Return Format

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

```
:MTES:SOUR CHAN1;ENAB 0;LOCK 1;:MTES:AMAS:SOUR CHAN1;UNIT DIV;
XDEL +2.50000000E-01;YDEL +2.50000000E-01;:MTES:SCAL:X1 +200.000E-06;
XDEL +400.000E-06;Y1 -3.00000E+00;Y2 +3.00000E+00;BIND 0;
:MTES:RMOD FOR;RMOD:TIME +1E+00;WAV +1.000E+03;SIGM +6.0E+00;
:MTES:RMOD:FACT:STOP 0;PRIN 0;SAVE 0
```

Example Code

```
' Mask testing commands example.
' ----------------------------------------------------------------------

Option Explicit

Public myMgr As VisaComLib.ResourceManager
Public myScope As VisaComLib.FormattedIO488
Public varQueryResult As Variant
Public strQueryResult As String

#If VBA7 Then
    Private Declare PtrSafe Sub Sleep Lib "kernel32" (ByVal dwMillisecon
ds As LongPtr)
```

```
#Else
    Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Lo
ng)
#End If

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("USB0::0x2A8D::0x4704::Unset::0::INSTR")
  myScope.IO.Clear   ' Clear the interface.
  myScope.IO.Timeout = 10000   ' Set I/O communication timeout.

  ' Make sure oscilloscope is running.
  myScope.WriteString ":RUN"

  ' Enable mask testing.
  myScope.WriteString ":MTESt:ENABle ON"

  ' Set mask test termination conditions.
  myScope.WriteString ":MTESt:RMODe SIGMa"
  myScope.WriteString ":MTESt:RMODe?"
  strQueryResult = myScope.ReadString
  Debug.Print "Mask test termination mode: " + strQueryResult

  myScope.WriteString ":MTESt:RMODe:SIGMa 4.2"
  myScope.WriteString ":MTESt:RMODe:SIGMa?"
  varQueryResult = myScope.ReadNumber
  Debug.Print "Mask test termination 'test sigma': " + _
      FormatNumber(varQueryResult)

  ' Use auto-mask to create mask.
  myScope.WriteString ":MTESt:AMASk:SOURce CHANnel1"
  myScope.WriteString ":MTESt:AMASk:SOURce?"
  strQueryResult = myScope.ReadString
  Debug.Print "Mask test auto-mask source: " + strQueryResult

  myScope.WriteString ":MTESt:AMASk:UNITs DIVisions"
  myScope.WriteString ":MTESt:AMASk:UNITs?"
  strQueryResult = myScope.ReadString
  Debug.Print "Mask test auto-mask units: " + strQueryResult

  myScope.WriteString ":MTESt:AMASk:XDELta 0.1"
  myScope.WriteString ":MTESt:AMASk:XDELta?"
  varQueryResult = myScope.ReadNumber
  Debug.Print "Mask test auto-mask X delta: " + _
      FormatNumber(varQueryResult)

  myScope.WriteString ":MTESt:AMASk:YDELta 0.1"
  myScope.WriteString ":MTESt:AMASk:YDELta?"
  varQueryResult = myScope.ReadNumber
  Debug.Print "Mask test auto-mask Y delta: " + _
      FormatNumber(varQueryResult)
```

```
  ' Enable "Auto Mask Created" event (bit 10, &H400)
  myScope.WriteString "*CLS"
  myScope.WriteString ":STATus:OPERation:MTESt:ENABle " + CStr(CInt("&H4
00"))

  ' Create mask.
  myScope.WriteString ":MTESt:AMASk:CREate"
  Debug.Print "Auto-mask created, mask test automatically enabled."

  ' Set up timeout variables.
  Dim lngTimeout As Long   ' Max millisecs to wait.
  Dim lngElapsed As Long
  lngTimeout = 60000   ' 60 seconds.

  ' Wait until mask is created.
  lngElapsed = 0
  Do While lngElapsed <= lngTimeout
    myScope.WriteString ":STATus:OPERation:CONDition?"
    varQueryResult = myScope.ReadNumber
    ' Operation Status Condition Register MTE bit (bit 9, &H200).
    If (varQueryResult And &H200) <> 0 Then
      Exit Do
    Else
      Sleep 100   ' Small wait to prevent excessive queries.
      lngElapsed = lngElapsed + 100
    End If
  Loop

  ' Look for RUN bit = stopped (mask test termination).
  lngElapsed = 0
  Do While lngElapsed <= lngTimeout
    myScope.WriteString ":STATus:OPERation:CONDition?"
    varQueryResult = myScope.ReadNumber
    ' Operation Status Condition Register RUN bit (bit 3, &H8).
    If (varQueryResult And &H8) = 0 Then
      Exit Do
    Else
      Sleep 100   ' Small wait to prevent excessive queries.
      lngElapsed = lngElapsed + 100
    End If
  Loop

  ' Get total waveforms, failed waveforms, and test time.
  myScope.WriteString ":MTESt:COUNt:WAVeforms?"
  strQueryResult = myScope.ReadString
  Debug.Print "Mask test total waveforms: " + strQueryResult

  myScope.WriteString ":MTESt:COUNt:FWAVeforms?"
  strQueryResult = myScope.ReadString
  Debug.Print "Mask test failed waveforms: " + strQueryResult

  myScope.WriteString ":MTESt:COUNt:TIME?"
  strQueryResult = myScope.ReadString
  Debug.Print "Mask test elapsed seconds: " + strQueryResult

  Exit Sub
```

```
VisaComError:
  MsgBox "VISA COM Error:" + vbCrLf + Err.Description

End Sub
```

:MTESt:ALL

(see page 1292)

Command Syntax

```
:MTESt:ALL <on_off>

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

The :MTESt:ALL command specifies the channel(s) that are included in the mask test:

- ON – All displayed analog channels are included in the mask test.
- OFF – Just the selected source channel is included in the test.

Query Syntax

```
:MTESt:ALL?
```

The :MTESt:ALL? query returns the current setting.

Return Format

```
<on_off><NL>

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

See Also

- "Introduction to :MTESt Commands" on page 633

:MTESt:AMASk:CREate

 (see page 1292)

Command Syntax `:MTESt:AMASk:CREate`

The :MTESt:AMASk:CREate command automatically constructs a mask around the current selected channel, using the tolerance parameters defined by the :MTESt:AMASk:XDELta, :MTESt:AMASk:YDELta, and :MTESt:AMASk:UNITs commands. The mask only encompasses the portion of the waveform visible on the display, so you must ensure that the waveform is acquired and displayed consistently to obtain repeatable results.

The :MTESt:SOURce command selects the channel and should be set before using this command.

See Also

- "Introduction to :MTESt Commands" on page 633
- ":MTESt:AMASk:XDELta" on page 641
- ":MTESt:AMASk:YDELta" on page 642
- ":MTESt:AMASk:UNITs" on page 640
- ":MTESt:AMASk:SOURce" on page 639
- ":MTESt:SOURce" on page 664

Example Code

- "Example Code" on page 633

:MTESt:AMASk:SOURce

N (see page 1292)

Command Syntax

```
:MTESt:AMASk:SOURce <source>

<source> ::= CHANnel<n>

<n> ::= 1 to (# analog channels) in NR1 format
```

The :MTESt:AMASk:SOURce command selects the source for the interpretation of the :MTESt:AMASk:XDELta and :MTESt:AMASk:YDELta parameters when :MTESt:AMASk:UNITs is set to CURRent.

When UNITs are CURRent, the XDELta and YDELta parameters are defined in terms of the channel units, as set by the :CHANnel<n>:UNITs command, of the selected source.

Suppose that UNITs are CURRent and that you set SOURce to CHANnel1, which is using units of volts. Then you can define AMASk:XDELta in terms of volts and AMASk:YDELta in terms of seconds.

This command is the same as the :MTESt:SOURce command.

Query Syntax

```
:MTESt:AMASk:SOURce?
```

The :MTESt:AMASk:SOURce? query returns the currently set source.

Return Format

```
<source> ::= CHAN<n>

<n> ::= 1 to (# analog channels) in NR1 format
```

See Also

- "Introduction to :MTESt Commands" on page 633
- ":MTESt:AMASk:XDELta" on page 641
- ":MTESt:AMASk:YDELta" on page 642
- ":MTESt:AMASk:UNITs" on page 640
- ":MTESt:SOURce" on page 664

Example Code

- "Example Code" on page 633

:MTESt:AMASk:UNITs

N (see page 1292)

Command Syntax

```
:MTESt:AMASk:UNITs <units>

<units> ::= {CURRent | DIVisions}
```

The :MTESt:AMASk:UNITs command alters the way the mask test subsystem interprets the tolerance parameters for automasking as defined by :MTESt:AMASk:XDELta and :MTESt:AMASk:YDELta commands.

- CURRent – the mask test subsystem uses the units as set by the :CHANnel<n>:UNITs command, usually time for ΔX and voltage for ΔY.
- DIVisions – the mask test subsystem uses the graticule as the measurement system, so tolerance settings are specified as parts of a screen division. The mask test subsystem maintains separate XDELta and YDELta settings for CURRent and DIVisions. Thus, XDELta and YDELta are not converted to new values when the UNITs setting is changed.

Query Syntax

```
:MTESt:AMASk:UNITs?
```

The :MTESt:AMASk:UNITs query returns the current measurement units setting for the mask test automask feature.

Return Format

```
<units><NL>

<units> ::= {CURR | DIV}
```

See Also

- "Introduction to :MTESt Commands" on page 633
- ":MTESt:AMASk:XDELta" on page 641
- ":MTESt:AMASk:YDELta" on page 642
- ":CHANnel<n>:UNITs" on page 297
- ":MTESt:AMASk:SOURce" on page 639
- ":MTESt:SOURce" on page 664

Example Code

- "Example Code" on page 633

:MTESt:AMASk:XDELta

(see page 1292)

Command Syntax

```
:MTESt:AMASk:XDELta <value>

<value> ::= X delta value in NR3 format
```

The :MTESt:AMASk:XDELta command sets the tolerance in the X direction around the waveform for the automasking feature. The absolute value of the tolerance will be added and subtracted to horizontal values of the waveform to determine the boundaries of the mask.

The horizontal tolerance value is interpreted based on the setting specified by the :MTESt:AMASk:UNITs command; thus, if you specify 250-E3, the setting for :MTESt:AMASk:UNITs is CURRent, and the current setting specifies time in the horizontal direction, the tolerance will be ±250 ms. If the setting for :MTESt:AMASk:UNITs is DIVisions, the same X delta value will set the tolerance to ±250 millidivisions, or 1/4 of a division.

Query Syntax

```
:MTESt:AMASk:XDELta?
```

The :MTEST:AMASk:XDELta? query returns the current setting of the ΔX tolerance for automasking. If your computer program will interpret this value, it should also request the current measurement system using the :MTESt:AMASk:UNITs query.

Return Format

```
<value><NL>

<value> ::= X delta value in NR3 format
```

See Also

- "Introduction to :MTESt Commands" on page 633
- ":MTESt:AMASk:UNITs" on page 640
- ":MTESt:AMASk:YDELta" on page 642
- ":MTESt:AMASk:SOURce" on page 639
- ":MTESt:SOURce" on page 664

Example Code

- "Example Code" on page 633

:MTESt:AMASk:YDELta

(see page 1292)

Command Syntax

```
:MTESt:AMASk:YDELta <value>

<value> ::= Y delta value in NR3 format
```

The :MTESt:AMASk:YDELta command sets the vertical tolerance around the waveform for the automasking feature. The absolute value of the tolerance will be added and subtracted to vertical values of the waveform to determine the boundaries of the mask.

The vertical tolerance value is interpreted based on the setting specified by the :MTESt:AMASk:UNITs command; thus, if you specify 250-E3, the setting for :MTESt:AMASk:UNITs is CURRent, and the current setting specifies voltage in the vertical direction, the tolerance will be ±250 mV. If the setting for :MTESt:AMASk:UNITs is DIVisions, the same Y delta value will set the tolerance to ±250 millidivisions, or 1/4 of a division.

Query Syntax

```
:MTESt:AMASk:YDELta?
```

The :MTESt:AMASk:YDELta? query returns the current setting of the ΔY tolerance for automasking. If your computer program will interpret this value, it should also request the current measurement system using the :MTESt:AMASk:UNITs query.

Return Format

```
<value><NL>

<value> ::= Y delta value in NR3 format
```

See Also

- "Introduction to :MTESt Commands" on page 633
- ":MTESt:AMASk:UNITs" on page 640
- ":MTESt:AMASk:XDELta" on page 641
- ":MTESt:AMASk:SOURce" on page 639
- ":MTESt:SOURce" on page 664

Example Code

- "Example Code" on page 633

:MTESt:COUNt:FWAVeforms?

(see page 1292)

Query Syntax

```
:MTESt:COUNt:FWAVeforms? [CHANnel<n>]

<n> ::= 1 to (# analog channels) in NR1 format
```

The :MTESt:COUNt:FWAVeforms? query returns the total number of failed waveforms in the current mask test run. This count is for all regions and all waveforms collected on the channel specified by the optional parameter or collected on the currently specified source channel (:MTESt:SOURce) if there is no parameter.

Return Format

```
<failed><NL>

<failed> ::= number of failed waveforms in NR1 format.
```

See Also

- "Introduction to :MTESt Commands" on page 633
- ":MTESt:COUNt:WAVeforms?" on page 646
- ":MTESt:COUNt:TIME?" on page 645
- ":MTESt:COUNt:RESet" on page 644
- ":MTESt:SOURce" on page 664

Example Code

- "Example Code" on page 633

:MTESt:COUNt:RESet

 (see page 1292)

Command Syntax `:MTESt:COUNt:RESet`

The :MTESt:COUNt:RESet command resets the mask statistics.

See Also

- "Introduction to :MTESt Commands" on page 633
- ":MTESt:COUNt:WAVeforms?" on page 646
- ":MTESt:COUNt:FWAVeforms?" on page 643
- ":MTESt:COUNt:TIME?" on page 645

:MTESt:COUNt:TIME?

(see page 1292)

Query Syntax

```
:MTESt:COUNt:TIME?
```

The :MTESt:COUNt:TIME? query returns the elapsed time in the current mask test run.

Return Format

```
<time><NL>

<time> ::= elapsed seconds in NR3 format.
```

See Also

- "Introduction to :MTESt Commands" on page 633
- ":MTESt:COUNt:WAVeforms?" on page 646
- ":MTESt:COUNt:FWAVeforms?" on page 643
- ":MTESt:COUNt:RESet" on page 644

Example Code

- "Example Code" on page 633

:MTESt:COUNt:WAVeforms?

(see page 1292)

Query Syntax

```
:MTESt:COUNt:WAVeforms?
```

The :MTESt:COUNt:WAVeforms? query returns the total number of waveforms acquired in the current mask test run.

Return Format

```
<count><NL>

<count> ::= number of waveforms in NR1 format.
```

See Also

- "Introduction to :MTESt Commands" on page 633
- ":MTESt:COUNt:FWAVeforms?" on page 643
- ":MTESt:COUNt:TIME?" on page 645
- ":MTESt:COUNt:RESet" on page 644

Example Code

- "Example Code" on page 633

:MTESt:DATA

(see page 1292)

Command Syntax

```
:MTESt:DATA <mask>

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

The :MTESt:DATA command loads a mask from binary block data.

Query Syntax

```
:MTESt:DATA?
```

The :MTESt:DATA? query returns a mask in binary block data format. The format for the data transmission is the # format defined in the IEEE 488.2 specification.

Return Format

```
<mask><NL>

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

See Also

- ":SAVE:MASK[:STARt]" on page 697
- ":RECall:MASK[:STARt]" on page 682

:MTESt:DELete

 (see page 1292)

Command Syntax `:MTESt:DELete`

The :MTESt:DELete command clears the currently loaded mask.

See Also

- "Introduction to :MTESt Commands" on page 633
- ":MTESt:AMASk:CREate" on page 638

:MTESt:ENABle

N (see page 1292)

Command Syntax

```
:MTESt:ENABle <on_off>
<on_off> ::= {{1 | ON} | {0 | OFF}}
```

The :MTESt:ENABle command enables or disables the mask test features.

- ON – Enables the mask test features.
- OFF – Disables the mask test features.

Query Syntax

```
:MTESt:ENABle?
```

The :MTESt:ENABle? query returns the current state of mask test features.

Return Format

```
<on_off><NL>
<on_off> ::= {1 | 0}
```

See Also

- "Introduction to :MTESt Commands" on page 633

:MTESt:LOCK

(see page 1292)

Command Syntax

```
:MTESt:LOCK <on_off>

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

The :MTESt:LOCK command enables or disables the mask lock feature:

- ON – Locks a mask to the SOURce. As the vertical or horizontal scaling or position of the SOURce changes, the mask is redrawn accordingly.
- OFF – The mask is static and does not move.

Query Syntax

```
:MTESt:LOCK?
```

The :MTESt:LOCK? query returns the current mask lock setting.

Return Format

```
<on_off><NL>

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

See Also

- "Introduction to :MTESt Commands" on page 633
- ":MTESt:SOURce" on page 664

:MTESt:RMODe

 (see page 1292)

Command Syntax

```
:MTESt:RMODe <rmode>

<rmode> ::= {FORever | SIGMa | TIME | WAVeforms}
```

The :MTESt:RMODe command specifies the termination conditions for the mask test:

- FORever – the mask test runs until it is turned off.
- SIGMa – the mask test runs until the Sigma level is reached. This level is set by the ":MTESt:RMODe:SIGMa" on page 656 command.
- TIME – the mask test runs for a fixed amount of time. The amount of time is set by the ":MTESt:RMODe:TIME" on page 657 command.
- WAVeforms – the mask test runs until a fixed number of waveforms are acquired. The number of waveforms is set by the ":MTESt:RMODe:WAVeforms" on page 658 command.

Query Syntax

```
:MTESt:RMODe?
```

The :MTESt:RMODe? query returns the currently set termination condition.

Return Format

```
<rmode><NL>

<rmode> ::= {FOR | SIGM | TIME | WAV}
```

See Also

- "Introduction to :MTESt Commands" on page 633
- ":MTESt:RMODe:SIGMa" on page 656
- ":MTESt:RMODe:TIME" on page 657
- ":MTESt:RMODe:WAVeforms" on page 658

Example Code

- "Example Code" on page 633

:MTESt:RMODe:FACTion:MEASure

(see page 1292)

Command Syntax

```
:MTESt:RMODe:FACTion:MEASure <on_off>

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

The :MTESt:RMODe:FACTion:MEASure command sets measuring only mask failures on or off.

When ON, measurements and measurement statistics run only on waveforms that contain a mask violation; passing waveforms do not affect measurements and measurement statistics.

This mode is not available when the acquisition mode is set to Averaging.

Query Syntax

```
:MTESt:RMODe:FACTion:MEASure?
```

The :MTESt:RMODe:FACTion:MEASure? query returns the current mask failure measure setting.

Return Format

```
<on_off><NL>

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

See Also

- "Introduction to :MTESt Commands" on page 633
- ":MTESt:RMODe:FACTion:PRINt" on page 653
- ":MTESt:RMODe:FACTion:SAVE" on page 654
- ":MTESt:RMODe:FACTion:STOP" on page 655

:MTESt:RMODe:FACTion:PRINt

(see page 1292)

Command Syntax

```
:MTESt:RMODe:FACTion:PRINt <on_off>

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

The :MTESt:RMODe:FACTion:PRINt command sets printing on mask failures on or off.

NOTE **Setting :MTESt:RMODe:FACTion:PRINt ON automatically sets :MTESt:RMODe:FACTion:SAVE OFF.**

See Chapter 20, ":HCOPy Commands," starting on page 469 for more information on setting the hardcopy device and formatting options.

Query Syntax

```
:MTESt:RMODe:FACTion:PRINt?
```

The :MTESt:RMODe:FACTion:PRINt? query returns the current mask failure print setting.

Return Format

```
<on_off><NL>

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

See Also

- "Introduction to :MTESt Commands" on page 633
- ":MTESt:RMODe:FACTion:MEASure" on page 652
- ":MTESt:RMODe:FACTion:SAVE" on page 654
- ":MTESt:RMODe:FACTion:STOP" on page 655

:MTESt:RMODe:FACTion:SAVE

(see page 1292)

Command Syntax

```
:MTESt:RMODe:FACTion:SAVE <on_off>

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

The :MTESt:RMODe:FACTion:SAVE command sets saving on mask failures on or off.

NOTE **Setting :MTESt:RMODe:FACTion:SAVE ON automatically sets :MTESt:RMODe:FACTion:PRINt OFF.**

See Chapter 29, ":SAVE Commands," starting on page 687 for more information on save options.

Query Syntax

```
:MTESt:RMODe:FACTion:SAVE?
```

The :MTESt:RMODe:FACTion:SAVE? query returns the current mask failure save setting.

Return Format

```
<on_off><NL>

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

See Also

- "Introduction to :MTESt Commands" on page 633
- ":MTESt:RMODe:FACTion:MEASure" on page 652
- ":MTESt:RMODe:FACTion:PRINt" on page 653
- ":MTESt:RMODe:FACTion:STOP" on page 655

:MTESt:RMODe:FACTion:STOP

(see page 1292)

Command Syntax

```
:MTESt:RMODe:FACTion:STOP <on_off>

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

The :MTESt:RMODe:FACTion:STOP command sets stopping on a mask failure on or off. When this setting is ON and a mask violation is detected, the mask test is stopped and the acquisition system is stopped.

Query Syntax

```
:MTESt:RMODe:FACTion:STOP?
```

The :MTESt:RMODe:FACTion:STOP? query returns the current mask failure stop setting.

Return Format

```
<on_off><NL>

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

See Also

- "Introduction to :MTESt Commands" on page 633
- ":MTESt:RMODe:FACTion:MEASure" on page 652
- ":MTESt:RMODe:FACTion:PRINt" on page 653
- ":MTESt:RMODe:FACTion:SAVE" on page 654

:MTESt:RMODe:SIGMa

(see page 1292)

Command Syntax

```
:MTESt:RMODe:SIGMa <level>

<level> ::= from 0.1 to 9.3 in NR3 format
```

When the :MTESt:RMODe command is set to SIGMa, the :MTESt:RMODe:SIGMa command sets the *test sigma* level to which a mask test runs. *Test sigma* is the best achievable process sigma, assuming no failures. (*Process sigma* is calculated using the number of failures per test.) The *test sigma* level indirectly specifies the number of waveforms that must be tested (in order to reach the sigma level).

Query Syntax

```
:MTESt:RMODe:SIGMa?
```

The :MTESt:RMODe:SIGMa? query returns the current Sigma level setting.

Return Format

```
<level><NL>

<level> ::= from 0.1 to 9.3 in NR3 format
```

See Also

- "Introduction to :MTESt Commands" on page 633
- ":MTESt:RMODe" on page 651

Example Code

- "Example Code" on page 633

:MTESt:RMODe:TIME

 (see page 1292)

Command Syntax

```
:MTESt:RMODe:TIME <seconds>

<seconds> ::= from 1 to 86400 in NR3 format
```

When the :MTESt:RMODe command is set to TIME, the :MTESt:RMODe:TIME command sets the number of seconds for a mask test to run.

Query Syntax

```
:MTESt:RMODe:TIME?
```

The :MTESt:RMODe:TIME? query returns the number of seconds currently set.

Return Format

```
<seconds><NL>

<seconds> ::= from 1 to 86400 in NR3 format
```

See Also

- "Introduction to :MTESt Commands" on page 633
- ":MTESt:RMODe" on page 651

:MTESt:RMODe:WAVeforms

N (see page 1292)

Command Syntax

```
:MTESt:RMODe:WAVeforms <count>

<count> ::= number of waveforms in NR1 format
            from 1 to 2,000,000,000
```

When the :MTESt:RMODe command is set to WAVeforms, the :MTESt:RMODe:WAVeforms command sets the number of waveform acquisitions that are mask tested.

Query Syntax

```
:MTESt:RMODe:WAVeforms?
```

The :MTESt:RMODe:WAVeforms? query returns the number of waveforms currently set.

Return Format

```
<count><NL>

<count> ::= number of waveforms in NR1 format
            from 1 to 2,000,000,000
```

See Also

- "Introduction to :MTESt Commands" on page 633
- ":MTESt:RMODe" on page 651

:MTESt:SCALe:BIND

(see page 1292)

Command Syntax

```
:MTESt:SCALe:BIND <on_off>

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

The :MTESt:SCALe:BIND command enables or disables Bind 1 & 0 Levels (Bind -1 & 0 Levels for inverted masks) control:

- ON –

  If the Bind 1 & 0 Levels control is enabled, the 1 Level and the 0 Level controls track each other. Adjusting either the 1 Level or the 0 Level control shifts the position of the mask up or down without changing its size.

  If the Bind -1 & 0 Levels control is enabled, the -1 Level and the 0 Level controls track each other. Adjusting either the -1 Level or the 0 Level control shifts the position of the mask up or down without changing its size.

- OFF –

  If the Bind 1 & 0 Levels control is disabled, adjusting either the 1 Level or the 0 Level control changes the vertical height of the mask.

  If the Bind -1 & 0 Levels control is disabled, adjusting either the -1 Level or the 0 Level control changes the vertical height of the mask.

Query Syntax

```
:MTESt:SCALe:BIND?
```

The :MTESt:SCALe:BIND? query returns the value of the Bind 1&0 control (Bind -1&0 for inverted masks).

Return Format

```
<on_off><NL>

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

See Also

- "Introduction to :MTESt Commands" on page 633
- ":MTESt:SCALe:X1" on page 660
- ":MTESt:SCALe:XDELta" on page 661
- ":MTESt:SCALe:Y1" on page 662
- ":MTESt:SCALe:Y2" on page 663

:MTESt:SCALe:X1

N (see page 1292)

Command Syntax

```
:MTESt:SCALe:X1 <x1_value>

<x1_value> ::= X1 value in NR3 format
```

The :MTESt:SCALe:X1 command defines where X=0 in the base coordinate system used for mask testing. The other X-coordinate is defined by the :MTESt:SCALe:XDELta command. Once the X1 and XDELta coordinates are set, all X values of vertices in the mask regions are defined with respect to this value, according to the equation:

$X = (X * \Delta X) + X1$

Thus, if you set X1 to 100 ms, and XDELta to 100 ms, an X value of 0.100 is a vertex at 110 ms.

The oscilloscope uses this equation to normalize vertices. This simplifies reprogramming to handle different data rates. For example, if you halve the period of the waveform of interest, you need only to adjust the XDELta value to set up the mask for the new waveform.

The X1 value is a time value specifying the location of the X1 coordinate, which will then be treated as X=0 for mask regions coordinates.

Query Syntax

```
:MTESt:SCALe:X1?
```

The :MTESt:SCALe:X1? query returns the current X1 coordinate setting.

Return Format

```
<x1_value><NL>

<x1_value> ::= X1 value in NR3 format
```

See Also

- "Introduction to :MTESt Commands" on page 633
- ":MTESt:SCALe:BIND" on page 659
- ":MTESt:SCALe:XDELta" on page 661
- ":MTESt:SCALe:Y1" on page 662
- ":MTESt:SCALe:Y2" on page 663

:MTESt:SCALe:XDELta

(see page 1292)

Command Syntax

```
:MTESt:SCALe:XDELta <xdelta_value>

<xdelta_value> ::= X delta value in NR3 format
```

The :MTESt:SCALe:XDELta command defines the position of the X2 marker with respect to the X1 marker. In the mask test coordinate system, the X1 marker defines where X=0; thus, the X2 marker defines where X=1.

Because all X vertices of the regions defined for mask testing are normalized with respect to X1 and ΔX, redefining ΔX also moves those vertices to stay in the same locations with respect to X1 and ΔX. Thus, in many applications, it is best if you define XDELta as a pulse width or bit period. Then, a change in data rate without corresponding changes in the waveform can easily be handled by changing ΔX.

The X-coordinate of polygon vertices is normalized using this equation:

```
X = (X * ΔX) + X1
```

The X delta value is a time value specifying the distance of the X2 marker with respect to the X1 marker.

For example, if the period of the waveform you wish to test is 1 ms, setting ΔX to 1 ms ensures that the waveform's period is between the X1 and X2 markers.

Query Syntax

```
:MTESt:SCALe:XDELta?
```

The :MTESt:SCALe:XDELta? query returns the current value of ΔX.

Return Format

```
<xdelta_value><NL>

<xdelta_value> ::= X delta value in NR3 format
```

See Also

- "Introduction to :MTESt Commands" on page 633
- ":MTESt:SCALe:BIND" on page 659
- ":MTESt:SCALe:X1" on page 660
- ":MTESt:SCALe:Y1" on page 662
- ":MTESt:SCALe:Y2" on page 663

:MTESt:SCALe:Y1

(see page 1292)

Command Syntax

```
:MTESt:SCALe:Y1 <y1_value>

<y1_value> ::= Y1 value in NR3 format
```

The :MTESt:SCALe:Y1 command defines where Y=0 in the coordinate system for mask testing. All Y values of vertices in the coordinate system are defined with respect to the boundaries set by SCALe:Y1 and SCALe:Y2 according to the equation:

```
Y = (Y * (Y2 - Y1)) + Y1
```

Thus, if you set Y1 to 100 mV, and Y2 to 1 V, a Y value of 0.100 in a vertex is at 190 mV.

The Y1 value is a voltage value specifying the point at which Y=0.

Query Syntax

```
:MTESt:SCALe:Y1?
```

The :MTESt:SCALe:Y1? query returns the current setting of the Y1 marker.

Return Format

```
<y1_value><NL>

<y1_value> ::= Y1 value in NR3 format
```

See Also

- "Introduction to :MTESt Commands" on page 633
- ":MTESt:SCALe:BIND" on page 659
- ":MTESt:SCALe:X1" on page 660
- ":MTESt:SCALe:XDELta" on page 661
- ":MTESt:SCALe:Y2" on page 663

:MTESt:SCALe:Y2

(see page 1292)

Command Syntax

```
:MTESt:SCALe:Y2 <y2_value>

<y2_value> ::= Y2 value in NR3 format
```

The :MTESt:SCALe:Y2 command defines the Y2 marker in the coordinate system for mask testing. All Y values of vertices in the coordinate system are defined with respect to the boundaries defined by SCALe:Y1 and SCALe:Y2 according to the following equation:

```
Y = (Y * (Y2 - Y1)) + Y1
```

Thus, if you set Y1 to 100 mV, and Y2 to 1 V, a Y value of 0.100 in a vertex is at 190 mV.

The Y2 value is a voltage value specifying the location of the Y2 marker.

Query Syntax

```
:MTESt:SCALe:Y2?
```

The :MTESt:SCALe:Y2? query returns the current setting of the Y2 marker.

Return Format

```
<y2_value><NL>

<y2_value> ::= Y2 value in NR3 format
```

See Also

- "Introduction to :MTESt Commands" on page 633
- ":MTESt:SCALe:BIND" on page 659
- ":MTESt:SCALe:X1" on page 660
- ":MTESt:SCALe:XDELta" on page 661
- ":MTESt:SCALe:Y1" on page 662

:MTESt:SOURce

(see page 1292)

Command Syntax

```
:MTESt:SOURce <source>

<source> ::= CHANnel<n>

<n> ::= 1 to (# analog channels) in NR1 format
```

The :MTESt:SOURce command selects the channel which is configured by the commands contained in a mask file when it is loaded.

Query Syntax

```
:MTESt:SOURce?
```

The :MTESt:SOURce? query returns the channel which is configured by the commands contained in the current mask file.

Return Format

```
<source><NL>

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

<n> ::= 1 to (# analog channels) in NR1 format
```

See Also

- "Introduction to :MTESt Commands" on page 633
- ":MTESt:AMASk:SOURce" on page 639

:MTESt:TITLe?

 (see page 1292)

Query Syntax `:MTESt:TITLe?`

The :MTESt:TITLe? query returns the mask title which is a quoted string of up to 128 characters. The title is displayed in the mask test dialog box and mask test tab when a mask file is loaded.

Return Format `<title><NL>`

`<title> ::= a quoted string of up to 128 ASCII characters.`

See Also

- "Introduction to :MTESt Commands" on page 633

27 :POD Commands

Control all oscilloscope functions associated with groups of digital channels. See "Introduction to :POD<n> Commands" on page 667.

Table 92 :POD<n> Commands Summary

Command	Query	Options and Query Returns
:POD<n>:DISPlay {{0 | OFF} | {1 | ON}} (see page 669)	:POD<n>:DISPlay? (see page 669)	{0 | 1} <n> ::= 1-2 in NR1 format
:POD<n>:NIBBle<n>:THReshold <value> (see page 670)	:POD<n>:NIBBle<n>:THReshold? (see page 670)	<n> ::= 1-2 in NR1 format <value> :: = {CMOS | ECL | TTL | <decimal>[<suffix>]} <decimal> ::= -8.00 to +8.00 in NR3 format [suffix] ::= {V | mV | uV }
:POD<n>:SIZE <value> (see page 672)	:POD<n>:SIZE? (see page 672)	<value> ::= {SMALl | MEDium | LARGe}
:POD<n>:THReshold <value> (see page 673)	:POD<n>:THReshold? (see page 673)	<n> ::= 1-2 in NR1 format <value> :: = {CMOS | ECL | TTL | <decimal>[<suffix>]} <decimal> ::= -8.00 to +8.00 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

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<n>:DISPlay

N (see page 1292)

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.

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" on page 667
- ":DIGital<d>:DISPlay" on page 314
- ":CHANnel<n>:DISPlay" on page 275
- ":VIEW" on page 227
- ":BLANk" on page 217
- ":STATus?" on page 224

:POD<n>:NIBBle<n>:THReshold

N (see page 1292)

Command Syntax

```
:POD<n>:NIBBle<n>:THReshold <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.

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

<decimal> ::= -8.00 to +8.00 in NR3 format

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

POD1 ::= D0-D7

POD2 ::= D8-D15
```

The :POD<n>:NIBBle<n>:THReshold command sets the logic threshold value for a nibble within a pod:

- Pod 1 has the digital channels 0 through 7.
  - Pod 1, nibble 1 has the digital channels 0 through 3.
  - Pod 1, nibble 2 has the digital channels 4 through 7.
- Pod 2 has the digital channels 8 through 15.
  - Pod 2, nibble 1 has the digital channels 8 through 11.
  - Pod 2, nibble 2 has the digital channels 12 through 15.

	POD2								POD1							
	NIBBle2				NIBBle1				NIBBle2				NIBBle1			
Digital channel	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

The threshold is used for triggering purposes and for displaying the digital data as high (above the threshold) or low (below the threshold).

The voltage values for the predefined thresholds are:

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

Query Syntax

```
:POD<n>:NIBBle<n>:THReshold?
```

The :POD<N>:NIBBle<N>:THReshold? query returns the threshold value for the specified pod and nibble.

Return Format

```
<threshold><NL>
```

```
<threshold> ::= Floating point number in NR3 format
```

See Also
- "Introduction to :POD<n> Commands" on page 667
- ":POD<n>:THReshold" on page 673
- ":DIGital<d>:THReshold" on page 318
- ":TRIGger[:EDGE]:LEVel" on page 1096

Example Code

This example sets the threshold to 1.8 volts for pod 2, nibble 2 (digital channels D15 through D12).

```
myScope.WriteString ":POD2:NIBBle2:THReshold 1.8"
```

:POD<n>:SIZE

(see page 1292)

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. Sizes are set for all pods. Therefore, if you set the size on pod 1 (for example), the same size is set on pod 2 as well.

Query Syntax

```
:POD<n>:SIZE?
```

The :POD<n>:SIZE? query returns the digital channels size setting.

Return Format

```
<size_value><NL>

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

See Also

- "Introduction to :POD<n> Commands" on page 667
- ":DIGital<d>:SIZE" on page 317
- ":DIGital:ORDer[:SET]" on page 313

:POD<n>:THReshold

N (see page 1292)

Command Syntax

```
:POD<n>:THReshold <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.

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

<decimal> ::= -8.00 to +8.00 in NR3 format

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

POD1 ::= D0-D7

POD2 ::= D8-D15
```

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).

The voltage values for the predefined thresholds are:

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

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" on page 667
- ":DIGital<d>:THReshold" on page 318
- ":TRIGger[:EDGE]:LEVel" on page 1096

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.

' Set channels 0-7 to CMOS threshold.
myScope.WriteString ":POD1:THReshold CMOS"
```

```
' Set channels 8-15 to 2.0 volts.
myScope.WriteString ":POD2:THReshold 2.0"

' Set external channel to TTL threshold (short form).
myScope.WriteString ":TRIG:LEV TTL,EXT"
```

See complete example programs at: Chapter 44, “Programming Examples,” starting on page 1301

28 :RECall Commands

Recall previously saved oscilloscope setups, reference waveforms, and masks.

Table 93 :RECall Commands Summary

Command	Query	Options and Query Returns
:RECall:ARBitrary:[STARt] [{<internal_fname> | <external_fname>}][, <column>][, <wavegen_id>] (see page 678)	n/a	<internal_fname> ::= quoted ASCII file name string beginning with "/User Files/" <external_fname> ::= quoted ASCII file name string beginning with "/USB1/" (or "/usb/"), "/USB2/", or "/USB3/" depending on the number of USB storage devices connected <column> ::= Column in CSV file to load. Column number starts from 1. <wavegen_id> ::= WGEN1
:RECall:DBC[:STARt] [<file_name>] [, <serialbus>] (see page 679)	n/a	<file_name> ::= quoted ASCII string If extension included in file name, it must be ".dbc". <serialbus> ::= {SBUS<n>} <n> ::= 1 to (# of serial bus) in NR1 format
:RECall:FILename <base_name> (see page 680)	:RECall:FILename? (see page 680)	<base_name> ::= quoted ASCII string

Table 93 :RECall Commands Summary (continued)

Command	Query	Options and Query Returns
:RECall:LDF[:STARt] [<file_name>] [, <serialbus>] (see page 681)	n/a	<file_name> ::= quoted ASCII string If extension included in file name, it must be ".ldf". <serialbus> ::= {SBUS<n>} <n> ::= 1 to (# of serial bus) in NR1 format
:RECall:MASK[:STARt] [{<internal_fname> | <external_fname>}] (see page 682)	n/a	<internal_fname> ::= quoted ASCII file name string beginning with "/User Files/" <external_fname> ::= quoted ASCII file name string beginning with "/USB1/" (or "/usb/"), "/USB2/", or "/USB3/" depending on the number of USB storage devices connected
:RECall:PWD <path_name> (see page 683)	:RECall:PWD? (see page 683)	<path_name> ::= quoted ASCII string
:RECall:SETup[:STARt] [{<internal_fname> | <external_fname>}] (see page 684)	n/a	<internal_fname> ::= quoted ASCII file name string beginning with "/User Files/" <external_fname> ::= quoted ASCII file name string beginning with "/USB1/" (or "/usb/"), "/USB2/", or "/USB3/" depending on the number of USB storage devices connected
:RECall:WMEMory<r>[:STARt] [<file_name> | <data>] (see page 685)	n/a	<r> ::= 1 to (# ref waveforms) in NR1 format <file_name> ::= quoted ASCII string If extension included in file name, it must be ".h5". <data> ::= binary block data in IEEE 488.2 # format

Introduction to :RECall Commands

The :RECall subsystem provides commands to recall previously saved oscilloscope setups, reference waveforms, and masks.

Reporting the Setup

Use :RECall? to query setup information for the RECall subsystem.

Return Format

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

```
:REC:FIL "scope_0"
```

Recalling Files From a USB Storage Device

When :RECall commands have a "quoted ASCII string" <file_name> parameter, you can recall files from a connected USB storage device. For example:

```
' To recall a setup file from a connected USB storage device:
myScope.WriteString ":RECall:SETup:STARt ""/usb/my_setup_file.scp"""
```

:RECall:ARBitrary[:STARt]

(see page 1292)

Command Syntax

```
:RECall:ARBitrary[:STARt]
   [{<internal_fname> | <external_fname>}][, <column>][, <wavegen_id>]

<internal_fname> ::= quoted ASCII file name string beginning
   with "/User Files/"

<external_fname> ::= quoted ASCII file name string beginning
   with "/USB1/" (or "/usb/"), "/USB2/", or "/USB3/" depending
   on the number of USB storage devices connected

<column> ::= Column in CSV file to load. Column number starts from 1.

<wavegen_id> ::= WGEN1 - specifies which wavegen
```

The :RECall:ARBitrary[:STARt] command recalls an arbitrary waveform.

NOTE If a file extension is provided as part of a specified <file_name>, it must be ".csv".

For internal locations, the <column> parameter is ignored.

For external (USB storage device) files, the column parameter is optional. If no <column> parameter is entered, and it is a 2-column file, the 2nd column (assumed to be voltage) is automatically be selected. If the <column> parameter is entered, and that column does not exist in the file, the operation fails.

When recalling arbitrary waveforms (from an external USB storage device) that were not saved from the oscilloscope, be aware that the oscilloscope uses a maximum of 8192 points for an arbitrary waveform. For more efficient recalls, make sure your arbitrary waveforms are 8192 points or less.

The <wavegen_id> parameter specifies which waveform generator to recall the arbitrary waveform into.

See Also

- "Introduction to :RECall Commands" on page 676
- ":RECall:FILename" on page 680
- ":RECall:PWD" on page 683
- ":SAVE:ARBitrary[:STARt]" on page 691
- "Commands Not Supported in Multiple Program Message Units" on page 1297

:RECall:DBC[:STARt]

(see page 1292)

Command Syntax

```
:RECall:DBC[:STARt] [<file_name>] [, <serialbus>]

<file_name> ::= quoted ASCII string

<serialbus> ::= {SBUS<n>}

<n> ::= 1 to (# of serial bus) in NR1 format
```

The :RECall:DBC[:STARt] command loads a CAN DBC (communication database) symbolic data file into the oscilloscope.

NOTE If a file extension is provided as part of a specified <file_name>, it must be ".dbc".

The <serialbus> parameter specifies which serial decode waveform the CAN symbolic data will be loaded for.

See Also

- "Introduction to :RECall Commands" on page 676
- ":RECall:FILename" on page 680
- ":SBUS<n>:CAN:TRIGger" on page 737
- ":SBUS<n>:CAN:TRIGger:SYMBolic:MESSage" on page 747
- ":SBUS<n>:CAN:TRIGger:SYMBolic:SIGNal" on page 748
- ":SBUS<n>:CAN:TRIGger:SYMBolic:VALue" on page 749
- ":SEARch:SERial:CAN:MODE" on page 848
- ":SEARch:SERial:CAN:SYMBolic:MESSage" on page 854
- ":SEARch:SERial:CAN:SYMBolic:SIGNal" on page 855
- ":SEARch:SERial:CAN:SYMBolic:VALue" on page 856
- "Commands Not Supported in Multiple Program Message Units" on page 1297

:RECall:FILename

(see page 1292)

Command Syntax

```
:RECall:FILename <base_name>
<base_name> ::= quoted ASCII string
```

The :RECall:FILename command specifies the source for any RECall operations.

NOTE This command specifies a file's base name only, without path information or an extension.

Query Syntax

```
:RECall:FILename?
```

The :RECall:FILename? query returns the current RECall filename.

Return Format

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

See Also

- "Introduction to :RECall Commands" on page 676
- ":RECall:SETup[:STARt]" on page 684
- ":SAVE:FILename" on page 692

:RECall:LDF[:STARt]

(see page 1292)

Command Syntax

```
:RECall:LDF[:STARt] [<file_name>] [, <serialbus>]
<file_name> ::= quoted ASCII string
<serialbus> ::= {SBUS<n>}
<n> ::= 1 to (# of serial bus) in NR1 format
```

The :RECall:LDF[:STARt] command loads a LIN description file (LDF) symbolic data file into the oscilloscope.

NOTE If a file extension is provided as part of a specified <file_name>, it must be ".ldf".

The <serialbus> parameter specifies which serial decode waveform the LIN symbolic data will be loaded for.

See Also

- "Introduction to :RECall Commands" on page 676
- ":RECall:FILename" on page 680
- ":SBUS<n>:LIN:TRIGger" on page 772
- ":SBUS<n>:LIN:TRIGger:SYMBolic:FRAMe" on page 778
- ":SBUS<n>:LIN:TRIGger:SYMBolic:SIGNal" on page 779
- ":SBUS<n>:LIN:TRIGger:SYMBolic:VALue" on page 780
- ":SEARch:SERial:LIN:MODE" on page 866
- ":SEARch:SERial:LIN:SYMBolic:FRAMe" on page 870
- ":SEARch:SERial:LIN:SYMBolic:SIGNal" on page 871
- ":SEARch:SERial:LIN:SYMBolic:VALue" on page 872
- "Commands Not Supported in Multiple Program Message Units" on page 1297

:RECall:MASK[:STARt]

(see page 1292)

Command Syntax

```
:RECall:MASK[:STARt] [{<internal_fname> | <external_fname>}]

<internal_fname> ::= quoted ASCII file name string beginning
   with "/User Files/"

<external_fname> ::= quoted ASCII file name string beginning
   with "/USB1/" (or "/usb/"), "/USB2/", or "/USB3/" depending
   on the number of USB storage devices connected
```

The :RECall:MASK[:STARt] command recalls a mask.

NOTE If a file extension is provided as part of a specified <file_name>, it must be ".msk".

If no filename parameter is included, the name specified by the :RECall:FILename command and the location specified by the :RECall:PWD command are used.

NOTE When a USB storage device is connected, the :RECall:PWD location becomes "/usb/". When all USB storage devices are disconnected, the :RECall:PWD location becomes "/User Files/".

See Also

- "Introduction to :RECall Commands" on page 676
- ":RECall:FILename" on page 680
- ":RECall:PWD" on page 683
- ":SAVE:MASK[:STARt]" on page 697
- ":MTESt:DATA" on page 647

:RECall:PWD

 (see page 1292)

Command Syntax

```
:RECall:PWD <path_name>

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

The :RECall:PWD command sets the present working directory for recall operations.

Query Syntax

```
:RECall:PWD?
```

The :RECall:PWD? query returns the currently set working directory for recall operations.

Return Format

```
<path_name><NL>

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

See Also

- "Introduction to :RECall Commands" on page 676
- ":SAVE:PWD" on page 699

:RECall:SETup[:STARt]

N (see page 1292)

Command Syntax

```
:RECall:SETup[:STARt] [{<internal_fname> | <external_fname>}]

<internal_fname> ::= quoted ASCII file name string beginning
   with "/User Files/"

<external_fname> ::= quoted ASCII file name string beginning
   with "/USB1/" (or "/usb/"), "/USB2/", or "/USB3/" depending
   on the number of USB storage devices connected
```

The :RECall:SETup[:STARt] command recalls an oscilloscope setup from a file.

NOTE If a file extension is provided as part of a specified <file_name>, it must be ".scp".

If no filename parameter is included, the name specified by the :RECall:FILename command and the location specified by the :RECall:PWD command are used.

NOTE When a USB storage device is connected, the :RECall:PWD location becomes "/usb/". When all USB storage devices are disconnected, the :RECall:PWD location becomes "/User Files/".

See Also

- "Introduction to :RECall Commands" on page 676
- ":RECall:FILename" on page 680
- ":RECall:PWD" on page 683
- ":SAVE:SETup[:STARt]" on page 707
- "*RCL (Recall)" on page 193

:RECall:WMEMory<r>[:STARt]

(see page 1292)

Command Syntax

```
:RECall:WMEMory<r>[:STARt] [<file_name> | <data>]

<r> ::= 1 to (# ref waveforms) in NR1 format

<file_name> ::= quoted ASCII string

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

The :RECall:WMEMory<r>[:STARt] command recalls a reference waveform.

NOTE If a file extension is provided as part of a specified <file_name>, it must be ".h5".

The <data> option lets you recall a reference waveform from a local file on the controller PC (instead of from a USB storage device connected to the oscilloscope). In this case, your remote program reads data from the local ".h5" format reference waveform file in the same way that setup files are restored to the oscilloscope (see ":SYSTem:SETup" on page 1055).

See Also

- "Introduction to :RECall Commands" on page 676
- ":RECall:FILename" on page 680
- ":SAVE:WMEMory[:STARt]" on page 714
- "Commands Not Supported in Multiple Program Message Units" on page 1297

29 :SAVE Commands

Save oscilloscope setups, screen images, and data. See "Introduction to :SAVE Commands" on page 690.

Table 94 :SAVE Commands Summary

Command	Query	Options and Query Returns
:SAVE:ARBitrary:[STARt] [{<internal_fname> | <external_fname>}][, <wavegen_id>] (see page 691)	n/a	<internal_fname> ::= quoted ASCII file name string beginning with "/User Files/" <external_fname> ::= quoted ASCII file name string beginning with "/USB1/" (or "/usb/"), "/USB2/", or "/USB3/" depending on the number of USB storage devices connected <wavegen_id> ::= WGEN1
:SAVE:FILename <base_name> (see page 692)	:SAVE:FILename? (see page 692)	<base_name> ::= quoted ASCII string
:SAVE:IMAGe[:STARt] [<file_name>] (see page 693)	n/a	<file_name> ::= quoted ASCII string
:SAVE:IMAGe:FACTors {{0 | OFF} | {1 | ON}} (see page 694)	:SAVE:IMAGe:FACTors? (see page 694)	{0 | 1}
:SAVE:IMAGe:FORMat <format> (see page 695)	:SAVE:IMAGe:FORMat? (see page 695)	<format> ::= {{BMP | BMP24bit} | PNG | NONE}
:SAVE:LISTer[:STARt] [<file_name>] (see page 696)	n/a	<file_name> ::= quoted ASCII string

Table 94 :SAVE Commands Summary (continued)

Command	Query	Options and Query Returns
:SAVE:MASK[:STARt] [{<internal_fname> | <external_fname>}] (see page 697)	n/a	<internal_fname> ::= quoted ASCII file name string beginning with "/User Files/" <external_fname> ::= quoted ASCII file name string beginning with "/USB1/" (or "/usb/"), "/USB2/", or "/USB3/" depending on the number of USB storage devices connected
:SAVE:MULTi[:STARt] [<file_name>] (see page 698)	n/a	<file_name> ::= quoted ASCII string
:SAVE:PWD <path_name> (see page 699)	:SAVE:PWD? (see page 699)	<path_name> ::= quoted ASCII string
:SAVE:RESults:[STARt] [<file_spec>] (see page 700)	n/a	<file_name> ::= quoted ASCII string
:SAVE:RESults:FORMat:CURSor {{0 | OFF} | {1 | ON}} (see page 701)	:SAVE:RESults:FORMat:CURSor? (see page 701)	{0 | 1}
:SAVE:RESults:FORMat:HISTogram {{0 | OFF} | {1 | ON}} (see page 702)	:SAVE:RESults:FORMat:HISTogram? (see page 702)	{0 | 1}
:SAVE:RESults:FORMat:MASK {{0 | OFF} | {1 | ON}} (see page 703)	:SAVE:RESults:FORMat:MASK? (see page 703)	{0 | 1}
:SAVE:RESults:FORMat:MEASurement {{0 | OFF} | {1 | ON}} (see page 704)	:SAVE:RESults:FORMat:MEASurement? (see page 704)	{0 | 1}
:SAVE:RESults:FORMat:SEARch {{0 | OFF} | {1 | ON}} (see page 705)	:SAVE:RESults:FORMat:SEARch? (see page 705)	{0 | 1}
:SAVE:RESults:FORMat:SEGMented {{0 | OFF} | {1 | ON}} (see page 706)	:SAVE:RESults:FORMat:SEGMented? (see page 706)	{0 | 1}

Table 94 :SAVE Commands Summary (continued)

Command	Query	Options and Query Returns
:SAVE:SETup[:STARt] [{<internal_fname> | <external_fname>}] (see page 707)	n/a	<internal_fname> ::= quoted ASCII file name string beginning with "/User Files/" <external_fname> ::= quoted ASCII file name string beginning with "/USB1/" (or "/usb/"), "/USB2/", or "/USB3/" depending on the number of USB storage devices connected
:SAVE:WAVeform[:STARt] [<file_name>] (see page 708)	n/a	<file_name> ::= quoted ASCII string
:SAVE:WAVeform:FORMat <format> (see page 709)	:SAVE:WAVeform:FORMat? (see page 709)	<format> ::= {ASCiixy | CSV | BINary | NONE}
:SAVE:WAVeform:LENGth <length> (see page 710)	:SAVE:WAVeform:LENGth? (see page 710)	<length> ::= 100 to max. length; an integer in NR1 format
:SAVE:WAVeform:LENGth:MAX {{0 | OFF} | {1 | ON}} (see page 711)	:SAVE:WAVeform:LENGth:MAX? (see page 711)	{0 | 1}
:SAVE:WAVeform:SEGMented <option> (see page 712)	:SAVE:WAVeform:SEGMented? (see page 712)	<option> ::= {ALL | CURRent}
:SAVE:WMEMory:SOURce <source> (see page 713)	:SAVE:WMEMory:SOURce? (see page 713)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format NOTE: Only ADD or SUBtract math operations can be saved as reference waveforms. <return_value> ::= <source>
:SAVE:WMEMory[:STARt] [<file_name>] (see page 714)	n/a	<file_name> ::= quoted ASCII string If extension included in file name, it must be ".h5".

Introduction to :SAVE Commands

The :SAVE subsystem provides commands to save oscilloscope setups, screen images, and data.

:SAV is an acceptable short form for :SAVE.

Reporting the Setup

Use :SAVE? to query setup information for the SAVE subsystem.

Return Format

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

```
:SAVE:FIL "scope_0";:SAVE:IMAG:FACT 0;FORM NONE;
:SAVE:PWD "/User Files/setups/";:SAVE:WAV:FORM NONE;LENG +2000;
SEGM CURR
```

Saving Files to a USB Storage Device

When :SAVE commands have a "quoted ASCII string" <file_name> parameter, you can save files to a connected USB storage device. For example:

```
' To save a setup file to a connected USB storage device:
myScope.WriteString ":SAVE:SETup:STARt ""/usb/my_setup_file.scp"""
```

:SAVE:ARBitrary[:STARt]

(see page 1292)

Command Syntax

```
:SAVE:ARBitrary[:STARt]
   [{<internal_fname> | <external_fname>}][, <wavegen_id>]

<internal_fname> ::= quoted ASCII file name string beginning
   with "/User Files/"

<external_fname> ::= quoted ASCII file name string beginning
   with "/USB1/" (or "/usb/"), "/USB2/", or "/USB3/" depending
   on the number of USB storage devices connected

<wavegen_id> ::= WGEN1
```

The :SAVE:ARBitrary[:STARt] command saves the current arbitrary waveform to an internal location or a file on a USB storage device.

The <wavegen_id> parameter specifies which waveform generator to save the arbitrary waveform from.

NOTE If a file extension is provided as part of a specified <file_name>, it must be ".csv".

If no filename parameter is included, the name specified by the :SAVE:FILename command and the location specified by the :SAVE:PWD command are used.

NOTE When a USB storage device is connected, the :SAVE:PWD location becomes "/usb/". When all USB storage devices are disconnected, the :SAVE:PWD location becomes "/User Files/".

See Also

- "Introduction to :SAVE Commands" on page 690
- ":SAVE:FILename" on page 692
- ":SAVE:PWD" on page 699
- ":RECall:ARBitrary[:STARt]" on page 678
- "Commands Not Supported in Multiple Program Message Units" on page 1297

:SAVE:FILename

(see page 1292)

Command Syntax

```
:SAVE:FILename <base_name>
<base_name> ::= quoted ASCII string
```

The :SAVE:FILename command specifies the source for any SAVE operations.

NOTE This command specifies a file's base name only, without path information or an extension.

Query Syntax

```
:SAVE:FILename?
```

The :SAVE:FILename? query returns the current SAVE filename.

Return Format

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

See Also

- "Introduction to :SAVE Commands" on page 690
- ":SAVE:IMAGe[:STARt]" on page 693
- ":SAVE:SETup[:STARt]" on page 707
- ":SAVE:WAVeform[:STARt]" on page 708
- ":SAVE:PWD" on page 699
- ":RECall:FILename" on page 680

:SAVE:IMAGe[:STARt]

(see page 1292)

Command Syntax

```
:SAVE:IMAGe[:STARt] [<file_name>]

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

The :SAVE:IMAGe[:STARt] command saves an image.

NOTE Be sure to set the :SAVE:IMAGe:FORMat before saving an image. If the format is NONE, the save image command will not succeed.

NOTE If a file extension is provided as part of a specified <file_name>, and it does not match the extension expected by the format specified in :SAVE:IMAGe:FORMat, the format will be changed if the extension is a valid image file extension.

NOTE If the extension ".bmp" is used and the current :SAVE:IMAGe:FORMat is not BMP or BMP8, the format will be changed to BMP.

See Also

- "Introduction to :SAVE Commands" on page 690
- ":SAVE:IMAGe:FACTors" on page 694
- ":SAVE:IMAGe:FORMat" on page 695
- ":SAVE:FILename" on page 692

:SAVE:IMAGe:FACTors

 (see page 1292)

Command Syntax

```
:SAVE:IMAGe:FACTors <factors>

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

The :SAVE:IMAGe:FACTors command controls whether the oscilloscope factors are output along with the image.

NOTE **Factors are written to a separate file with the same path and base name but with the ".txt" extension.**

Query Syntax

```
:SAVE:IMAGe:FACTors?
```

The :SAVE:IMAGe:FACTors? query returns a flag indicating whether oscilloscope factors are output along with the image.

Return Format

```
<factors><NL>

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

See Also

- "Introduction to :SAVE Commands" on page 690
- ":SAVE:IMAGe[:STARt]" on page 693
- ":SAVE:IMAGe:FORMat" on page 695

:SAVE:IMAGe:FORMat

 (see page 1292)

Command Syntax

```
:SAVE:IMAGe:FORMat <format>

<format> ::= {{BMP | BMP24bit} | PNG}
```

The :SAVE:IMAGe:FORMat command sets the image format type.

Query Syntax

```
:SAVE:IMAGe:FORMat?
```

The :SAVE:IMAGe:FORMat? query returns the selected image format type.

Return Format

```
<format><NL>

<format> ::= {BMP | PNG | NONE}
```

When NONE is returned, it indicates that a waveform data file format is currently selected.

See Also

- "Introduction to :SAVE Commands" on page 690
- ":SAVE:IMAGe[:STARt]" on page 693
- ":SAVE:IMAGe:FACTors" on page 694
- ":SAVE:WAVeform:FORMat" on page 709

:SAVE:LISTer[:STARt]

(see page 1292)

Command Syntax

```
:SAVE:LISTer[:STARt] [<file_name>]
<file_name> ::= quoted ASCII string
```

The :SAVE:LISTer[:STARt] command saves the Lister display data to a file.

NOTE If a file extension is provided as part of a specified <file_name>, it must be ".csv".

See Also

- "Introduction to :SAVE Commands" on page 690
- ":SAVE:FILename" on page 692
- Chapter 22, ":LISTer Commands," starting on page 489

:SAVE:MASK[:STARt]

 (see page 1292)

Command Syntax

```
:SAVE:MASK[:STARt] [{<internal_fname> | <external_fname>}]

<internal_fname> ::= quoted ASCII file name string beginning
   with "/User Files/"

<external_fname> ::= quoted ASCII file name string beginning
   with "/USB1/" (or "/usb/"), "/USB2/", or "/USB3/" depending
   on the number of USB storage devices connected
```

The :SAVE:MASK[:STARt] command saves a mask.

NOTE If a file extension is provided as part of a specified <file_name>, it must be ".msk".

If no filename parameter is included, the name specified by the :SAVE:FILename command and the location specified by the :SAVE:PWD command are used.

NOTE When a USB storage device is connected, the :SAVE:PWD location becomes "/usb/". When all USB storage devices are disconnected, the :SAVE:PWD location becomes "/User Files/".

See Also

- "Introduction to :SAVE Commands" on page 690
- ":SAVE:FILename" on page 692
- ":SAVE:PWD" on page 699
- ":RECall:MASK[:STARt]" on page 682
- ":MTESt:DATA" on page 647

:SAVE:MULTi[:STARt]

 (see page 1292)

Command Syntax

```
:SAVE:MULTi[:STARt] [<file_name>]

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

The :SAVE:MULTi[:STARt] command saves multi-channel waveform data to a file. This file can be opened by the N8900A Infiniium Offline oscilloscope analysis software.

NOTE If a file extension is provided as part of a specified <file_name>, it must be ".h5".

See Also

- "Introduction to :SAVE Commands" on page 690
- ":SAVE:FILename" on page 692
- ":SAVE:PWD" on page 699

:SAVE:PWD

 (see page 1292)

Command Syntax

```
:SAVE:PWD <path_name>

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

The :SAVE:PWD command sets the present working directory for save operations.

Query Syntax

```
:SAVE:PWD?
```

The :SAVE:PWD? query returns the currently set working directory for save operations.

Return Format

```
<path_name><NL>

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

See Also

- "Introduction to :SAVE Commands" on page 690
- ":SAVE:FILename" on page 692
- ":RECall:PWD" on page 683

:SAVE:RESults[:STARt]

(see page 1292)

Command Syntax

```
:SAVE:RESults[:STARt] [<file_spec>]

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

The :SAVE:RESults[:STARt] command saves analysis results to a comma-separated values (*.csv) file on a USB storage device.

Use the :SAVE:RESults:FORMat commands to specify the analysis types whose results are saved to the file.

When multiple types of analysis results are selected, they are all saved to the same file and separated by a blank line.

See Also

- ":SAVE:RESults:FORMat:CURSor" on page 701
- ":SAVE:RESults:FORMat:MASK" on page 703
- ":SAVE:RESults:FORMat:MEASurement" on page 704
- ":SAVE:RESults:FORMat:SEARch" on page 705
- ":SAVE:RESults:FORMat:SEGMented" on page 706

:SAVE:RESults:FORMat:CURSor

N (see page 1292)

Command Syntax `:SAVE:RESults:FORMat:CURSor {{0 | OFF} | {1 | ON}}`

The :SAVE:RESults:FORMat:CURSor command specifies whether cursor values will be included when analysis results are saved.

Analysis results are saved using the :SAVE:RESults:[STARt] command.

Other :SAVE:RESults:FORMat commands specify whether other types of analysis results are also saved.

When multiple types of analysis results are saved, they are all saved to the same file and separated by a blank line.

Query Syntax `:SAVE:RESults:FORMat:CURSor?`

The :SAVE:RESults:FORMat:CURSor? query returns whether cursor values will be included when analysis results are saved.

Return Format `<off_on><NL>`

`{0 | 1}`

See Also

- ":SAVE:RESults[:STARt]" on page 700
- ":SAVE:RESults:FORMat:MASK" on page 703
- ":SAVE:RESults:FORMat:MEASurement" on page 704
- ":SAVE:RESults:FORMat:SEARch" on page 705
- ":SAVE:RESults:FORMat:SEGMented" on page 706

:SAVE:RESults:FORMat:HISTogram

(see page 1292)

Command Syntax

```
:SAVE:RESults:FORMat:HISTogram {{0 | OFF} | {1 | ON}}
```

The :SAVE:RESults:FORMat:HISTogram command specifies whether histogram statistics will be included when analysis results are saved.

Analysis results are saved using the :SAVE:RESults:[STARt] command.

Other :SAVE:RESults:FORMat commands specify whether other types of analysis results are also saved.

When multiple types of analysis results are saved, they are all saved to the same file and separated by a blank line.

Query Syntax

```
:SAVE:RESults:FORMat:HISTogram?
```

The :SAVE:RESults:FORMat:HISTogram? query returns whether histogram statistics will be included when analysis results are saved.

Return Format

```
<off_on><NL>
```

```
{0 | 1}
```

See Also

- ":SAVE:RESults[:STARt]" on page 700
- ":SAVE:RESults:FORMat:CURSor" on page 701
- ":SAVE:RESults:FORMat:MASK" on page 703
- ":SAVE:RESults:FORMat:MEASurement" on page 704
- ":SAVE:RESults:FORMat:SEARch" on page 705
- ":SAVE:RESults:FORMat:SEGMented" on page 706

:SAVE:RESults:FORMat:MASK

(see page 1292)

Command Syntax

```
:SAVE:RESults:FORMat:MASK {{0 | OFF} | {1 | ON}}
```

The :SAVE:RESults:FORMat:MASK command specifies whether mask statistics will be included when analysis results are saved.

Analysis results are saved using the :SAVE:RESults:[STARt] command.

Other :SAVE:RESults:FORMat commands specify whether other types of analysis results are also saved.

When multiple types of analysis results are saved, they are all saved to the same file and separated by a blank line.

Query Syntax

```
:SAVE:RESults:FORMat:MASK?
```

The :SAVE:RESults:FORMat:MASK? query returns whether mask statistics will be included when analysis results are saved.

Return Format

```
<off_on><NL>
```

```
{0 | 1}
```

See Also

- ":SAVE:RESults[:STARt]" on page 700
- ":SAVE:RESults:FORMat:CURSor" on page 701
- ":SAVE:RESults:FORMat:MEASurement" on page 704
- ":SAVE:RESults:FORMat:SEARch" on page 705
- ":SAVE:RESults:FORMat:SEGMented" on page 706

:SAVE:RESults:FORMat:MEASurement

(see page 1292)

Command Syntax

```
:SAVE:RESults:FORMat:MEASurement {{0 | OFF} | {1 | ON}}
```

The :SAVE:RESults:FORMat:MEASurement command specifies whether measurement results will be included when analysis results are saved.

Analysis results are saved using the :SAVE:RESults:[STARt] command.

Other :SAVE:RESults:FORMat commands specify whether other types of analysis results are also saved.

When multiple types of analysis results are saved, they are all saved to the same file and separated by a blank line.

Query Syntax

```
:SAVE:RESults:FORMat:MEASurement?
```

The :SAVE:RESults:FORMat:MEASurement? query returns whether measurement results will be included when analysis results are saved.

Return Format

```
<off_on><NL>
```

```
{0 | 1}
```

See Also

- ":SAVE:RESults[:STARt]" on page 700
- ":SAVE:RESults:FORMat:CURSor" on page 701
- ":SAVE:RESults:FORMat:MASK" on page 703
- ":SAVE:RESults:FORMat:SEARch" on page 705
- ":SAVE:RESults:FORMat:SEGMented" on page 706

:SAVE:RESults:FORMat:SEARch

(see page 1292)

Command Syntax

```
:SAVE:RESults:FORMat:SEARch {{0 | OFF} | {1 | ON}}
```

The :SAVE:RESults:FORMat:SEARch command specifies whether found search event times will be included when analysis results are saved.

Analysis results are saved using the :SAVE:RESults:[STARt] command.

Other :SAVE:RESults:FORMat commands specify whether other types of analysis results are also saved.

When multiple types of analysis results are saved, they are all saved to the same file and separated by a blank line.

Query Syntax

```
:SAVE:RESults:FORMat:SEARch?
```

The :SAVE:RESults:FORMat:SEARch? query returns whether found search event times will be included when analysis results are saved.

Return Format

```
<off_on><NL>
```

```
{0 | 1}
```

See Also

- ":SAVE:RESults[:STARt]" on page 700
- ":SAVE:RESults:FORMat:CURSor" on page 701
- ":SAVE:RESults:FORMat:MASK" on page 703
- ":SAVE:RESults:FORMat:MEASurement" on page 704
- ":SAVE:RESults:FORMat:SEGMented" on page 706

:SAVE:RESults:FORMat:SEGMented

(see page 1292)

Command Syntax `:SAVE:RESults:FORMat:SEGMented {{0 | OFF} | {1 | ON}}`

The :SAVE:RESults:FORMat:SEGMented command specifies whether segmented memory acquisition times will be included when analysis results are saved.

Analysis results are saved using the :SAVE:RESults:[STARt] command.

Other :SAVE:RESults:FORMat commands specify whether other types of analysis results are also saved.

When multiple types of analysis results are saved, they are all saved to the same file and separated by a blank line.

Query Syntax `:SAVE:RESults:FORMat:SEGMented?`

The :SAVE:RESults:FORMat:SEGMented? query returns whether segmented memory acquisition times will be included when analysis results are saved.

Return Format `<off_on><NL>`

`{0 | 1}`

See Also

- ":SAVE:RESults[:STARt]" on page 700
- ":SAVE:RESults:FORMat:CURSor" on page 701
- ":SAVE:RESults:FORMat:MASK" on page 703
- ":SAVE:RESults:FORMat:MEASurement" on page 704
- ":SAVE:RESults:FORMat:SEARch" on page 705

:SAVE:SETup[:STARt]

(see page 1292)

Command Syntax

```
:SAVE:SETup[:STARt] [{<internal_fname> | <external_fname>}]

<internal_fname> ::= quoted ASCII file name string beginning
   with "/User Files/"

<external_fname> ::= quoted ASCII file name string beginning
   with "/USB1/" (or "/usb/"), "/USB2/", or "/USB3/" depending
   on the number of USB storage devices connected
```

The :SAVE:SETup[:STARt] command saves the oscilloscope setup to a file.

NOTE If a file extension is provided as part of a specified file name, it must be ".scp".

If no filename parameter is included, the name specified by the :SAVE:FILename command and the location specified by the :SAVE:PWD command are used.

NOTE When a USB storage device is connected, the :SAVE:PWD location becomes "/usb/". When all USB storage devices are disconnected, the :SAVE:PWD location becomes "/User Files/".

See Also

- "Introduction to :SAVE Commands" on page 690
- ":SAVE:FILename" on page 692
- ":SAVE:PWD" on page 699
- ":RECall:SETup[:STARt]" on page 684
- "*SAV (Save)" on page 197

:SAVE:WAVeform[:STARt]

(see page 1292)

Command Syntax

```
:SAVE:WAVeform[:STARt] [<file_name>]

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

The :SAVE:WAVeform[:STARt] command saves oscilloscope waveform data to a file.

NOTE Be sure to set the :SAVE:WAVeform:FORMat before saving waveform data. If the format is NONE, the save waveform command will not succeed.

NOTE If a file extension is provided as part of a specified <file_name>, and it does not match the extension expected by the format specified in :SAVE:WAVeform:FORMat, the format will be changed if the extension is a valid waveform file extension.

See Also

- "Introduction to :SAVE Commands" on page 690
- ":SAVE:WAVeform:FORMat" on page 709
- ":SAVE:WAVeform:LENGth" on page 710
- ":SAVE:FILename" on page 692
- ":RECall:SETup[:STARt]" on page 684

:SAVE:WAVeform:FORMat

N (see page 1292)

Command Syntax

```
:SAVE:WAVeform:FORMat <format>

<format> ::= {ASCiixy | CSV | BINary}
```

The :SAVE:WAVeform:FORMat command sets the waveform data format type:

- ASCiixy – creates comma-separated value files for each analog channel that is displayed (turned on). The proper file extension for this format is ".csv".
- CSV – creates one comma-separated value file that contains information for all analog channels that are displayed (turned on). The proper file extension for this format is ".csv".
- BINary – creates an oscilloscope binary data format file. See the *User's Guide* for a description of this format. The proper file extension for this format is ".bin".

Query Syntax

```
:SAVE:WAVeform:FORMat?
```

The :SAVE:WAVeform:FORMat? query returns the selected waveform data format type.

Return Format

```
<format><NL>

<format> ::= {ASC | CSV | BIN | NONE}
```

When NONE is returned, it indicates that an image file format is currently selected.

See Also

- "Introduction to :SAVE Commands" on page 690
- ":SAVE:WAVeform[:STARt]" on page 708
- ":SAVE:WAVeform:LENGth" on page 710
- ":SAVE:IMAGe:FORMat" on page 695

:SAVE:WAVeform:LENGth

(see page 1292)

Command Syntax

```
:SAVE:WAVeform:LENGth <length>

<length> ::= 100 to max. length; an integer in NR1 format
```

When the :SAVE:WAVeform:LENGth:MAX setting is OFF, the :SAVE:WAVeform:LENGth command sets the waveform data length (that is, the number of points saved).

When the :SAVE:WAVeform:LENGth:MAX setting is ON, the :SAVE:WAVeform:LENGth setting has no effect.

Query Syntax

```
:SAVE:WAVeform:LENGth?
```

The :SAVE:WAVeform:LENGth? query returns the current waveform data length setting.

Return Format

```
<length><NL>

<length> ::= 100 to max. length; an integer in NR1 format
```

See Also

- "Introduction to :SAVE Commands" on page 690
- ":SAVE:WAVeform:LENGth:MAX" on page 711
- ":SAVE:WAVeform[:STARt]" on page 708
- ":WAVeform:POINts" on page 1156
- ":SAVE:WAVeform:FORMat" on page 709

:SAVE:WAVeform:LENGth:MAX

(see page 1292)

Command Syntax

```
:SAVE:WAVeform:LENGth:MAX <setting>

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

The :SAVE:WAVeform:LENGth:MAX command specifies whether maximum number of waveform data points is saved.

When OFF, the :SAVE:WAVeform:LENGth command specifies the number of waveform data points saved.

Query Syntax

```
:SAVE:WAVeform:LENGth:MAX?
```

The :SAVE:WAVeform:LENGth:MAX? query returns the current setting.

Return Format

```
<setting><NL>

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

See Also

- "Introduction to :SAVE Commands" on page 690
- ":SAVE:WAVeform[:STARt]" on page 708
- ":SAVE:WAVeform:LENGth" on page 710

:SAVE:WAVeform:SEGMented

(see page 1292)

Command Syntax

```
:SAVE:WAVeform:SEGMented <option>

<option> ::= {ALL | CURRent}
```

When segmented memory is used for acquisitions, the :SAVE:WAVeform:SEGMented command specifies which segments are included when the waveform is saved:

- ALL – all acquired segments are saved.
- CURRent – only the currently selected segment is saved.

Query Syntax

```
:SAVE:WAVeform:SEGMented?
```

The :SAVE:WAVeform:SEGMented? query returns the current segmented waveform save option setting.

Return Format

```
<option><NL>

<option> ::= {ALL | CURR}
```

See Also

- "Introduction to :SAVE Commands" on page 690
- ":SAVE:WAVeform[:STARt]" on page 708
- ":SAVE:WAVeform:FORMat" on page 709
- ":SAVE:WAVeform:LENGth" on page 710

:SAVE:WMEMory:SOURce

N (see page 1292)

Command Syntax

```
:SAVE:WMEMory:SOURce <source>

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

<n> ::= 1 to (# analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :SAVE:WMEMory:SOURce command selects the source to be saved as a reference waveform file.

NOTE Only ADD or SUBtract math operations can be saved as reference waveforms.

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

Query Syntax

```
:SAVE:WMEMory:SOURce?
```

The :SAVE:WMEMory:SOURce? query returns the source to be saved as a reference waveform file.

Return Format

```
<source><NL>

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

See Also

- "Introduction to :SAVE Commands" on page 690
- ":SAVE:WMEMory[:STARt]" on page 714
- ":RECall:WMEMory<r>[:STARt]" on page 685

:SAVE:WMEMory[:STARt]

(see page 1292)

Command Syntax

```
:SAVE:WMEMory[:STARt] [<file_name>]

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

The :SAVE:WMEMory[:STARt] command saves oscilloscope waveform data to a reference waveform file.

NOTE If a file extension is provided as part of a specified <file_name>, it must be ".h5".

See Also

- "Introduction to :SAVE Commands" on page 690
- ":SAVE:WMEMory:SOURce" on page 713
- ":RECall:WMEMory<r>[:STARt]" on page 685

30 :SBUS<n> Commands

Control the modes and parameters for each serial bus decode/trigger type. See:

Introduction to :SBUS<n> Commands

The :SBUS subsystem commands control the serial decode bus viewing, mode, and other options.

NOTE **These commands are only valid on oscilloscope models when a serial decode option has been licensed.**

The following serial bus decode/trigger types are available (see ":TRIGger:MODE" on page 1091).

- **CAN (Controller Area Network) triggering** – will trigger on CAN 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. You can trigger on CAN data and identifier patterns and you can set the bit sample point.
- **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.You can trigger on Sync Break, Frame IDs, or Frame IDs and Data.

- **SPI (Serial Peripheral Interface) triggering** — consists of connecting the oscilloscope to a clock, data (MOSI or MISO), 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 64 bits long.
- **UART/RS-232 triggering** — lets you trigger on RS-232 serial data.

NOTE Two SPI buses cannot be decoded on both SBUS1 and SBUS2 at the same time.

Reporting the Setup

Use :SBUS<n>? to query setup information for the :SBUS<n> subsystem.

Return Format

The following is a sample response from the :SBUS1? query. In this case, the query was issued following a *RST command.

```
:SBUS1:DISP 0;MODE IIC;:SBUS1:IIC:ASIZ BIT7;:SBUS1:IIC:TRIG:TYPE STAR;
QUAL EQU;:SBUS1:IIC:SOUR:CLOC CHAN1;DATA CHAN2;
:SBUS1:IIC:TRIG:PATT:ADDR -1;DATA -1;DATA2 -1
```

General :SBUS<n> Commands

Table 95 General :SBUS<n> Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:DISPlay {{0 | OFF} | {1 | ON}} (see page 718)	:SBUS<n>:DISPlay? (see page 718)	{0 | 1}
:SBUS<n>:MODE <mode> (see page 719)	:SBUS<n>:MODE? (see page 719)	<mode> ::= {CAN | IIC | LIN | SPI | UART}

:SBUS<n>:DISPlay

N (see page 1292)

Command Syntax

```
:SBUS<n>:DISPlay <display>

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

The :SBUS<n>:DISPlay command turns displaying of the serial decode bus on or off.

NOTE This command is only valid when a serial decode option has been licensed.

NOTE Two SPI buses cannot be decoded on both SBUS1 and SBUS2 at the same time.

Query Syntax

```
:SBUS<n>:DISPlay?
```

The :SBUS<n>:DISPlay? query returns the current display setting of the serial decode bus.

Return Format

```
<display><NL>

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

Errors

- "-241, Hardware missing" on page 1247

See Also

- "Introduction to :SBUS<n> Commands" on page 715
- ":CHANnel<n>:DISPlay" on page 275
- ":DIGital<d>:DISPlay" on page 314
- ":POD<n>:DISPlay" on page 669
- ":VIEW" on page 227
- ":BLANk" on page 217
- ":STATus?" on page 224

:SBUS<n>:MODE

(see page 1292)

Command Syntax

```
:SBUS<n>:MODE <mode>
<mode> ::= {CAN | IIC | LIN | SPI | UART}
```

The :SBUS<n>:MODE command determines the decode mode for the serial bus.

NOTE This command is only valid when a serial decode option has been licensed.

Query Syntax

```
:SBUS<n>:MODE?
```

The :SBUS<n>:MODE? query returns the current serial bus decode mode setting.

Return Format

```
<mode><NL>
<mode> ::= {CAN | IIC | LIN | SPI | UART | NONE}
```

Errors

- "-241, Hardware missing" on page 1247

See Also

- "Introduction to :SBUS<n> Commands" on page 715
- ":SBUS<n>:CAN Commands" on page 720
- ":SBUS<n>:IIC Commands" on page 753
- ":SBUS<n>:LIN Commands" on page 763
- ":SBUS<n>:SPI Commands" on page 781
- ":SBUS<n>:UART Commands" on page 798

:SBUS<n>:CAN Commands

NOTE These commands are valid when the CAN and LIN serial decode license has been enabled.

Table 96 :SBUS<n>:CAN Commands Summary

Command	Query	Options and Query Returns
n/a	:SBUS<n>:CAN:COUNt:ERRor? (see page 723)	<frame_count> ::= integer in NR1 format
n/a	:SBUS<n>:CAN:COUNt:OVERload? (see page 724)	<frame_count> ::= 0 in NR1 format
:SBUS<n>:CAN:COUNt:RESet (see page 725)	n/a	n/a
n/a	:SBUS<n>:CAN:COUNt:SPEC? (see page 726)	<spec_error_count> ::= integer in NR1 format
n/a	:SBUS<n>:CAN:COUNt:TOTal? (see page 727)	<frame_count> ::= integer in NR1 format
n/a	:SBUS<n>:CAN:COUNt:UTILization? (see page 728)	<percent> ::= floating-point in NR3 format
:SBUS<n>:CAN:DISPlay <type> (see page 729)	:SBUS<n>:CAN:DISPlay? (see page 729)	<type> ::= {HEXadecimal | SYMBolic}
:SBUS<n>:CAN:FDSPoint <value> (see page 730)	:SBUS<n>:CAN:FDSPoint? (see page 730)	<value> ::= even numbered percentages from 30 to 90 in NR3 format.
:SBUS<n>:CAN:SAMPlepoint <percent> (see page 731)	:SBUS<n>:CAN:SAMPlepoint? (see page 731)	<percent> ::= 30.0 to 90.0 in NR3 format
:SBUS<n>:CAN:SIGNal:BAUDrate <baudrate> (see page 732)	:SBUS<n>:CAN:SIGNal:BAUDrate? (see page 732)	<baudrate> ::= integer from 10000 to 4000000 in 100 b/s increments, or 5000000
:SBUS<n>:CAN:SIGNal:DEFinition <value> (see page 733)	:SBUS<n>:CAN:SIGNal:DEFinition? (see page 733)	<value> ::= {CANH | CANL | RX | TX | DIFFerential | DIFL | DIFH}
:SBUS<n>:CAN:SIGNal:FDBaudrate <baudrate> (see page 734)	:SBUS<n>:CAN:SIGNal:FDBaudrate? (see page 734)	<baudrate> ::= integer from 10000 to 10000000 in 100 b/s increments.
:SBUS<n>:CAN:SIGNal:XLBaudrate <baudrate> (see page 735)	:SBUS<n>:CAN:SIGNal:XLBaudrate? (see page 735)	<baudrate> ::= integer from 10000 to 20000000 in 100 b/s increments.

Table 96 :SBUS<n>:CAN Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:CAN:SOURce <source> (see page 736)	:SBUS<n>:CAN:SOURce? (see page 736)	<source> ::= {CHANnel<n> | DIGital<d> |} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:SBUS<n>:CAN:TRIGger <condition> (see page 737)	:SBUS<n>:CAN:TRIGger? (see page 738)	<condition> ::= {SOF | EOF | IDData | DATA | IDRemote | IDEither | ERRor | ACKerror | FORMerror | STUFferror | CRCerror | SPECerror | ALLerrors | BRSBit | CRCDbit | EBActive | EBPassive | OVERload | MESSage | MSIGnal}
:SBUS<n>:CAN:TRIGger:IDFilter {{0 | OFF} | {1 | ON}} (see page 740)	:SBUS<n>:CAN:TRIGger:IDFilter? (see page 740)	{0 | 1}
:SBUS<n>:CAN:TRIGger:PATTern:DATA <string> (see page 741)	:SBUS<n>:CAN:TRIGger:PATTern:DATA? (see page 741)	<string> ::= "nn...n" where n ::= {0 | 1 | X | $} <string ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $}
:SBUS<n>:CAN:TRIGger:PATTern:DATA:DLC <dlc> (see page 742)	:SBUS<n>:CAN:TRIGger:PATTern:DATA:DLC? (see page 742)	<dlc> ::= integer between -1 (don't care) and 64, in NR1 format.
:SBUS<n>:CAN:TRIGger:PATTern:DATA:LENGth <length> (see page 743)	:SBUS<n>:CAN:TRIGger:PATTern:DATA:LENGth? (see page 743)	<length> ::= integer from 1 to 8 in NR1 format
:SBUS<n>:CAN:TRIGger:PATTern:DATA:STARt <start> (see page 744)	:SBUS<n>:CAN:TRIGger:PATTern:DATA:STARt? (see page 744)	<start> ::= integer between 0 and 63, in NR1 format.
:SBUS<n>:CAN:TRIGger:PATTern:ID <string> (see page 745)	:SBUS<n>:CAN:TRIGger:PATTern:ID? (see page 745)	<string> ::= "nn...n" where n ::= {0 | 1 | X | $} <string ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $}
:SBUS<n>:CAN:TRIGger:PATTern:ID:MODE <value> (see page 746)	:SBUS<n>:CAN:TRIGger:PATTern:ID:MODE? (see page 746)	<value> ::= {STANdard | EXTended}
:SBUS<n>:CAN:TRIGger:SYMBolic:MESSage <name> (see page 747)	:SBUS<n>:CAN:TRIGger:SYMBolic:MESSage? (see page 747)	<name> ::= quoted ASCII string

Table 96 :SBUS<n>:CAN Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:CAN:TRIGger:SYMBolic:SIGNal <name> (see page 748)	:SBUS<n>:CAN:TRIGger:SYMBolic:SIGNal? (see page 748)	<name> ::= quoted ASCII string
:SBUS<n>:CAN:TRIGger:SYMBolic:VALue <data> (see page 749)	:SBUS<n>:CAN:TRIGger:SYMBolic:VALue? (see page 749)	<data> ::= value in NR3 format
:SBUS<n>:CAN:TRIGger:TYPE <type> (see page 750)	:SBUS<n>:CAN:TRIGger:TYPE? (see page 750)	<type> ::= {STANdard | FD | XL}
:SBUS<n>:CAN:TYPE <type> (see page 751)	:SBUS<n>:CAN:TYPE? (see page 751)	<type> ::= {STANdard | FD | XFASt | XSIC}
:SBUS<n>:CAN:XLSPoint <value> (see page 752)	:SBUS<n>:CAN:XLSPoint? (see page 752)	<value> ::= even numbered percentages from 30 to 90 in NR3 format.

:SBUS<n>:CAN:COUNt:ERRor?

N (see page 1292)

Query Syntax `:SBUS<n>:CAN:COUNt:ERRor?`

Returns the error frame count.

Return Format `<frame_count><NL>`

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

Errors

- "-241, Hardware missing" on page 1247

See Also

- ":SBUS<n>:CAN:COUNt:RESet" on page 725
- "Introduction to :SBUS<n> Commands" on page 715
- ":SBUS<n>:MODE" on page 719
- ":SBUS<n>:CAN Commands" on page 720

:SBUS<n>:CAN:COUNt:OVERload?

(see page 1292)

Query Syntax `:SBUS<n>:CAN:COUNt:OVERload?`

Returns the overload frame count.

Return Format `<frame_count><NL>`

`<frame_count> ::= 0 in NR1 format`

Errors

- "-241, Hardware missing" on page 1247

See Also

- ":SBUS<n>:CAN:COUNt:RESet" on page 725
- "Introduction to :SBUS<n> Commands" on page 715
- ":SBUS<n>:MODE" on page 719
- ":SBUS<n>:CAN Commands" on page 720

:SBUS<n>:CAN:COUNt:RESet

N (see page 1292)

Command Syntax `:SBUS<n>:CAN:COUNt:RESet`

Resets the frame counters.

Errors

- "-241, Hardware missing" on page 1247

See Also

- ":SBUS<n>:CAN:COUNt:ERRor?" on page 723
- ":SBUS<n>:CAN:COUNt:OVERload?" on page 724
- ":SBUS<n>:CAN:COUNt:TOTal?" on page 727
- ":SBUS<n>:CAN:COUNt:UTILization?" on page 728
- "Introduction to :SBUS<n> Commands" on page 715
- ":SBUS<n>:MODE" on page 719
- ":SBUS<n>:CAN Commands" on page 720

:SBUS<n>:CAN:COUNt:SPEC?

N (see page 1292)

Query Syntax `:SBUS<n>:CAN:COUNt:SPEC?`

Returns the Spec error (Ack + Form + Stuff + CRC errors) count.

Return Format `<spec_error_count><NL>`

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

Errors

- "-241, Hardware missing" on page 1247

See Also

- ":SBUS<n>:CAN:COUNt:RESet" on page 725
- "Introduction to :SBUS<n> Commands" on page 715
- ":SBUS<n>:MODE" on page 719
- ":SBUS<n>:CAN Commands" on page 720

:SBUS<n>:CAN:COUNt:TOTal?

N (see page 1292)

Query Syntax `:SBUS<n>:CAN:COUNt:TOTal?`

Returns the total frame count.

Return Format `<frame_count><NL>`

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

Errors

- "-241, Hardware missing" on page 1247

See Also

- ":SBUS<n>:CAN:COUNt:RESet" on page 725
- "Introduction to :SBUS<n> Commands" on page 715
- ":SBUS<n>:MODE" on page 719
- ":SBUS<n>:CAN Commands" on page 720

:SBUS<n>:CAN:COUNt:UTILization?

N (see page 1292)

Query Syntax `:SBUS<n>:CAN:COUNt:UTILization?`

Returns the percent utilization.

Return Format `<percent><NL>`

`<percent> ::= floating-point in NR3 format`

Errors

- "-241, Hardware missing" on page 1247

See Also

- ":SBUS<n>:CAN:COUNt:RESet" on page 725
- "Introduction to :SBUS<n> Commands" on page 715
- ":SBUS<n>:MODE" on page 719
- ":SBUS<n>:CAN Commands" on page 720

:SBUS<n>:CAN:DISPlay

N (see page 1292)

Command Syntax

```
:SBUS<n>:CAN:DISPlay <type>

<type> ::= {HEXadecimal | SYMBolic}
```

The :SBUS<n>:CAN:DISPlay command specifies, when CAN symbolic data is loaded into the oscilloscope, whether symbolic values (from the DBC file) or hexadecimal values are displayed in the decode waveform and the Lister window.

Query Syntax

```
:SBUS<n>:CAN:DISPlay?
```

The :SBUS<n>:CAN:DISPlay? query returns the CAN decode display type.

Return Format

```
<type><NL>

<type> ::= {HEX | SYMB}
```

See Also

- ":RECall:DBC[:STARt]" on page 679

:SBUS<n>:CAN:FDSPoint

 (see page 1292)

Command Syntax `:SBUS<n>:CAN:FDSPoint <value>`

`<value> ::= even numbered percentages from 30 to 90 in NR3 format.`

The :SBUS<n>:CAN:FDSPoint 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 `:SBUS<n>:CAN:FDSPoint?`

The :SBUS<n>:CAN:FDSPoint? query returns the current CAN FD sample point setting.

Return Format `<value><NL>`

`<value> ::= even numbered percentages from 30 to 90 in NR3 format.`

See Also

- ":SBUS<n>:CAN:SIGNal:FDBaudrate" on page 734
- ":SBUS<n>:CAN:TYPE" on page 751

:SBUS<n>:CAN:SAMPlepoint

N (see page 1292)

Command Syntax

```
:SBUS<n>:CAN:SAMPlepoint <percent>

<percent><NL>

<percent> ::= 30.0 to 90.0 in NR3 format
```

The :SBUS<n>: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

```
:SBUS<n>:CAN:SAMPlepoint?
```

The :SBUS<n>:CAN:SAMPlepoint? query returns the current CAN sample point setting.

Return Format

```
<percent><NL>

<percent> ::= 30.0 to 90.0 in NR3 format
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":SBUS<n>:MODE" on page 719
- ":SBUS<n>:CAN:TRIGger" on page 737

:SBUS<n>:CAN:SIGNal:BAUDrate

N (see page 1292)

Command Syntax

```
:SBUS<n>:CAN:SIGNal:BAUDrate <baudrate>

<baudrate> ::= integer from 10000 to 4000000 in 100 b/s increments,
               or 5000000
```

The :SBUS<n>:CAN:SIGNal:BAUDrate command sets the standard baud rate of the CAN signal from 10 kb/s to 4 Mb/s in 100 b/s increments. If you enter a baud rate that is not divisible by 100 b/s, the baud rate is set to the nearest baud rate divisible by 100 b/s.

You can also set the baud rate of the CAN signal to 5 Mb/s. Fractional baud rates between 4 Mb/s and 5 Mb/s are not allowed.

If the baud rate you select does not match the system baud rate, false triggers may occur.

Query Syntax

```
:SBUS<n>:CAN:SIGNal:BAUDrate?
```

The :SBUS<n>:CAN:SIGNal:BAUDrate? query returns the current CAN baud rate setting.

Return Format

```
<baudrate><NL>

<baudrate> ::= integer from 10000 to 4000000 in 100 b/s increments,
               or 5000000
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":SBUS<n>:MODE" on page 719
- ":SBUS<n>:CAN:TRIGger" on page 737
- ":SBUS<n>:CAN:SIGNal:DEFinition" on page 733
- ":SBUS<n>:CAN:SOURce" on page 736

:SBUS<n>:CAN:SIGNal:DEFinition

N (see page 1292)

Command Syntax

```
:SBUS<n>:CAN:SIGNal:DEFinition <value>

<value> ::= {CANH | CANL | RX | TX | DIFFerential | DIFL | DIFH}
```

The :SBUS<n>:CAN:SIGNal:DEFinition command sets the CAN signal type when :SBUS<n>:CAN:TRIGger is set to SOF (start of frame). These signals can be set to:

Dominant high signals:

- CANH – the actual CAN_H differential bus signal.
- DIFH – the CAN differential (H-L) bus signal connected to an analog source channel using a differential probe.

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.
- DIFL – the CAN differential (L-H) bus signal connected to an analog source channel using a differential probe.
- DIFFerential – the CAN differential bus signal connected to an analog source channel using a differential probe. This is the same as DIFL.

Query Syntax

```
:SBUS<n>:CAN:SIGNal:DEFinition?
```

The :SBUS<n>:CAN:SIGNal:DEFinition? query returns the current CAN signal type.

Return Format

```
<value><NL>

<value> ::= {CANH | CANL | RX | TX | DIFL | DIFH}
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":SBUS<n>:MODE" on page 719
- ":SBUS<n>:CAN:SIGNal:BAUDrate" on page 732
- ":SBUS<n>:CAN:SOURce" on page 736
- ":SBUS<n>:CAN:TRIGger" on page 737

:SBUS<n>:CAN:SIGNal:FDBaudrate

N (see page 1292)

Command Syntax

```
:SBUS<n>:CAN:SIGNal:FDBaudrate <baudrate>

<baudrate> ::= integer from 10000 to 10000000 in 100 b/s increments.
```

The :SBUS<n>:CAN:SIGNal:FDBaudrate command sets the CAN FD baud rate from 10 kb/s to 10 Mb/s in 100 b/s increments. If you enter a baud rate that is not divisible by 100 b/s, the baud rate is set to the nearest baud rate divisible by 100 b/s.

For CAN FD, both the standard rate settings (see :SBUS<n>:CAN:SIGNal:BAUDrate) and the FD rate settings must be set correctly; otherwise, false triggers may occur.

Query Syntax

```
:SBUS<n>:CAN:SIGNal:FDBaudrate?
```

The :SBUS<n>:CAN:SIGNal:FDBaudrate? query returns the current CAN FD baud rate setting.

Return Format

```
<baudrate><NL>

<baudrate> ::= integer from 10000 to 10000000 in 100 b/s increments.
```

See Also

- ":SBUS<n>:CAN:FDSPoint" on page 730
- ":SBUS<n>:CAN:SIGNal:BAUDrate" on page 732
- ":SBUS<n>:CAN:TYPE" on page 751

:SBUS<n>:CAN:SIGNal:XLBaudrate

N (see page 1292)

Command Syntax

```
:SBUS<n>:CAN:SIGNal:XLBaudrate <baudrate>

<baudrate> ::= integer from 10000 to 20000000 in 100 b/s increments.
```

The :SBUS<n>:CAN:SIGNal:XLBaudrate command sets the CAN XL baud rate from 10 kb/s to 20 Mb/s in 100 b/s increments. If you enter a baud rate that is not divisible by 100 b/s, the baud rate is set to the nearest baud rate divisible by 100 b/s.

For CAN XL, both the standard rate settings (see :SBUS<n>:CAN:SIGNal:BAUDrate), the FD rate settings, and the XL rate settings must be set correctly; otherwise, false triggers may occur.

Query Syntax

```
:SBUS<n>:CAN:SIGNal:XLBaudrate?
```

The :SBUS<n>:CAN:SIGNal:XLBaudrate? query returns the current CAN XL baud rate setting.

Return Format

```
<baudrate><NL>

<baudrate> ::= integer from 10000 to 20000000 in 100 b/s increments.
```

See Also

- ":SBUS<n>:CAN:XLSPoint" on page 752
- ":SBUS<n>:CAN:SIGNal:BAUDrate" on page 732
- ":SBUS<n>:CAN:SIGNal:FDBaudrate" on page 734
- ":SBUS<n>:CAN:TYPE" on page 751

:SBUS<n>:CAN:SOURce

N (see page 1292)

Command Syntax

```
:SBUS<n>:CAN:SOURce <source>

<source> ::= {CHANnel<n> | DIGital<d>}

<n> ::= 1 to (# analog channels) in NR1 format

<d> ::= 0 to (# digital channels - 1) in NR1 format
```

The :SBUS<n>:CAN:SOURce command sets the source for the CAN signal.

Query Syntax

```
:SBUS<n>:CAN:SOURce?
```

The :SBUS<n>:CAN:SOURce? query returns the current source for the CAN signal.

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":SBUS<n>:MODE" on page 719
- ":SBUS<n>:CAN:TRIGger" on page 737
- ":SBUS<n>:CAN:SIGNal:DEFinition" on page 733

:SBUS<n>:CAN:TRIGger

N (see page 1292)

Command Syntax

```
:SBUS<n>:CAN:TRIGger <condition>

<condition> ::= {SOF | EOF | IDData | DATA | IDRemote
    | IDEither | ERRor | ACKerror | FORMerror | STUFferror | CRCerror
    | SPECerror | ALLerrors | BRSBit | CRCDbit | EBActive | EBPassive
    | OVERload | MESSage | MSIGnal}
```

The :SBUS<n>:CAN:TRIGger command sets the CAN trigger on condition:

Condition	Front-panel name	Description	Filter by ID*
SOF	SOF - Start of Frame	Triggers at the start bit for both data and overload frames.	
EOF	EOF - End of Frame	Triggers at the end of any frame.	X
IDEither	Frame ID	Triggers on any standard CAN (data or remote) or CAN FD frame at the end of the 11- or 29-bit ID field.	
IDData	Data Frame ID (non-FD)	Triggers on standard CAN data frames at the end of the 11- or 29-bit ID field.	
DATA	Data Frame ID and Data (non-FD)	With the STANdard trigger type (see :SBUS<n>:CAN:TRIGger:TYPE), triggers on any standard CAN data frame at the end of the last data byte defined in the trigger. The DLC of the packet must must match the number of bytes specified. With the FD or XL trigger types (see :SBUS<n>:CAN:TRIGger:TYPE), triggers on CAN FD frames at the end of the last data byte defined in the trigger. You can trigger on up to 8 bytes of data anywhere within the CAN FD data, which can be up to 64 bytes long.	
IDRemote	Remote Frame ID	Triggers on standard CAN remote frames at the end of the 11- or 29-bit ID field.	
ERRor	Error Frame	Triggers after 6 consecutive 0s while in a data frame, at the EOF.	X
ACKerror	Acknowledge Error	Triggers on the acknowledge bit if the polarity is incorrect.	X
FORMerror	Form Error	Triggers on reserved bit errors.	X

Condition	Front-panel name	Description	Filter by ID*
STUFferror	Stuff Error	Triggers on 6 consecutive 1s or 6 consecutive 0s, while in a non-error or non overload frame.	X
CRCerror	CRC Field Error	Triggers when the calculated CRC does not match the transmitted CRC. In addition, for FD frames, will also trigger if the Stuff Count is in error.	X
SPECerror	Spec Error (Ack or Form or Stuff or CRC)	Triggers on Ack, Form, Stuff, or CRC errors.	X
ALLerrors	All Errors	Triggers on all Spec errors and error frames.	X
BRSBit	BRS Bit (FD)	Triggers on the BRS bit of CAN FD frames.	X
CRCDbit	CRC Delimiter Bit (FD)	Triggers on the CRC delimiter bit in CAN FD frames.	X
EBActive	ESI Bit Active (FD)	Triggers on the ESI bit if set active.	X
EBPassive	ESI Bit Passive (FD)	Triggers on the ESI bit if set passive.	X
OVERload	Overload Frame	Triggers on an overload frame.	
MESSage	Message	Triggers on a symbolic message.	
MSIGnal	Message and Signal (non-FD)	With the STANdard trigger type (see :SBUS<n>:CAN:TRIGger:TYPE), triggers on a symbolic message and a signal value. With the FD or XL trigger types (see :SBUS<n>:CAN:TRIGger:TYPE), triggers on a symbolic message and a signal value, limited to the first 8 bytes of FD data.	
* Filtering by CAN IDs is available for these trigger conditions (see :SBUS<n>:CAN:TRIGger:IDFilter).			

CAN Id specification is set by the :SBUS<n>:CAN:TRIGger:PATTern:ID and:SBUS<n>:CAN:TRIGger:PATTern:ID:MODE commands.

CAN Data specification is set by the :SBUS<n>:CAN:TRIGger:PATTern:DATA command.

CAN Data Length Code is set by the :SBUS<n>:CAN:TRIGger:PATTern:DATA:LENGth command.

Query Syntax `:SBUS<n>:CAN:TRIGger?`

The :SBUS<n>:CAN:TRIGger? query returns the current CAN trigger on condition.

Return Format `<condition><NL>`

```
<condition> ::= {SOF | EOF | IDD | DATA | IDR | IDE | ERR | ACK
   | FORM | STUF | CRC | SPEC | ALL | BRSB | CRCD | EBA | EBP | OVER
   | MESS | MSIG}
```

Errors

- "-241, Hardware missing" on page 1247

See Also

- ":SBUS<n>:CAN:TRIGger:TYPE" on page 750
- "Introduction to :SBUS<n> Commands" on page 715
- ":SBUS<n>:MODE" on page 719
- ":SBUS<n>:CAN:TRIGger:PATTern:DATA" on page 741
- ":SBUS<n>:CAN:TRIGger:PATTern:DATA:LENGth" on page 743
- ":SBUS<n>:CAN:TRIGger:PATTern:ID" on page 745
- ":SBUS<n>:CAN:TRIGger:PATTern:ID:MODE" on page 746
- ":SBUS<n>:CAN:TRIGger:IDFilter" on page 740
- ":SBUS<n>:CAN:SIGNal:DEFinition" on page 733
- ":SBUS<n>:CAN:SOURce" on page 736
- ":RECall:DBC[:STARt]" on page 679
- ":SBUS<n>:CAN:TRIGger:SYMBolic:MESSage" on page 747
- ":SBUS<n>:CAN:TRIGger:SYMBolic:SIGNal" on page 748
- ":SBUS<n>:CAN:TRIGger:SYMBolic:VALue" on page 749

:SBUS<n>:CAN:TRIGger:IDFilter

N (see page 1292)

Command Syntax `:SBUS<n>:CAN:TRIGger:IDFilter {{0 | OFF} | {1 | ON}}`

The :SBUS<n>:CAN:TRIGger:IDFilter command specifies, in certain error and bit trigger modes, whether triggers are filtered by CAN IDs.

Query Syntax `:SBUS<n>:CAN:TRIGger:IDFilter?`

The :SBUS<n>:CAN:TRIGger:IDFilter? query returns the CAN trigger ID filter setting.

Return Format `<setting><NL>`

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

See Also

- ":SBUS<n>:CAN:TRIGger" on page 737

:SBUS<n>:CAN:TRIGger:PATTern:DATA

N (see page 1292)

Command Syntax

```
:SBUS<n>:CAN:TRIGger:PATTern:DATA <string>

<string> ::= "nn...n" where n ::= {0 | 1 | X | $}

<string ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $}
```

The :SBUS<n>:CAN:TRIGger:PATTern:DATA command defines the CAN data pattern resource according to the string parameter. This pattern, along with the data length (set by the :SBUS<n>:CAN:TRIGger:PATTern:DATA:LENGth command), control the data pattern searched for in each CAN message.

If the string parameter starts with "0x", it is a hexadecimal string made up of hexadecimal and X (don't care) characters; otherwise, it is a binary string made up of 0, 1, and X (don't care) characters.

NOTE

If more bits are sent for <string> than specified by the :SBUS<n>:CAN:TRIGger:PATTern:DATA:LENGth command, the most significant bits will be truncated. If the data length is changed after the <string> is programmed, the added or deleted bits will be added to or deleted from the least significant bits.

Query Syntax

```
:SBUS<n>:CAN:TRIGger:PATTern:DATA?
```

The :SBUS<n>:CAN:TRIGger:PATTern:DATA? query returns the current settings of the specified CAN data pattern resource in the binary string format.

Return Format

```
<string><NL> in nondecimal format
```

Errors

- "-241, Hardware missing" on page 1247

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":SBUS<n>:CAN:TRIGger:PATTern:DATA:LENGth" on page 743
- ":SBUS<n>:CAN:TRIGger:PATTern:ID" on page 745

:SBUS<n>:CAN:TRIGger:PATTern:DATA:DLC

N (see page 1292)

Command Syntax

```
:SBUS<n>:CAN:TRIGger:PATTern:DATA:DLC <dlc>

<dlc> ::= integer between -1 (don't care) and 64, in NR1 format.
```

The :SBUS<n>:CAN:TRIGger:PATTern:DATA:DLC command specifies the DLC value to be used in the CAN FD data trigger mode. A specific valid FD value can be specified, or -1 can be specified to indicate "don't care".

Query Syntax

```
:SBUS<n>:CAN:TRIGger:PATTern:DATA:DLC?
```

The :SBUS<n>:CAN:TRIGger:PATTern:DATA:DLC? query returns the currently set DLC value.

Return Format

```
<dlc><NL>

<dlc> ::= integer between -1 (don't care) and 64, in NR1 format.
```

See Also

- ":SBUS<n>:CAN:TRIGger:PATTern:DATA" on page 741

:SBUS<n>:CAN:TRIGger:PATTern:DATA:LENGth

N (see page 1292)

Command Syntax

```
:SBUS<n>:CAN:TRIGger:PATTern:DATA:LENGth <length>

<length> ::= integer from 1 to 8 in NR1 format
```

The :SBUS<n>:CAN:TRIGger: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 1 bytes to 8 bytes (64 bits). The value for these bytes is set by the :SBUS<n>:CAN:TRIGger:PATTern:DATA command.

Query Syntax

```
:SBUS<n>:CAN:TRIGger:PATTern:DATA:LENGth?
```

The :SBUS<n>:CAN:TRIGger: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

- "-241, Hardware missing" on page 1247

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":SBUS<n>:CAN:TRIGger:PATTern:DATA" on page 741
- ":SBUS<n>:CAN:SOURce" on page 736

:SBUS<n>:CAN:TRIGger:PATTern:DATA:STARt

N (see page 1292)

Command Syntax

```
:SBUS<n>:CAN:TRIGger:PATTern:DATA:STARt <start>

<start> ::= integer between 0 and 63, in NR1 format.
```

The :SBUS<n>:CAN:TRIGger:PATTern:DATA:STARt command specifies the starting byte position for CAN FD data triggers.

CAN FD frames can have up to 64 bytes of data. You can trigger on up to 8 bytes of data. The starting byte position setting lets you trigger on data anywhere within the frame.

Query Syntax

```
:SBUS<n>:CAN:TRIGger:PATTern:DATA:STARt?
```

The :SBUS<n>:CAN:TRIGger:PATTern:DATA:STARt? query returns the starting byte position setting.

Return Format

```
<start><NL>

<start> ::= integer between 0 and 63, in NR1 format.
```

See Also

- ":SBUS<n>:CAN:TRIGger:PATTern:DATA" on page 741
- ":SBUS<n>:CAN:TRIGger:PATTern:DATA:LENGth" on page 743

:SBUS<n>:CAN:TRIGger:PATTern:ID

N (see page 1292)

Command Syntax

```
:SBUS<n>:CAN:TRIGger:PATTern:ID <string>

<string> ::= "nn...n" where n ::= {0 | 1 | X | $}

<string ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $}
```

The :SBUS<n>:CAN:TRIGger:PATTern:ID command defines the CAN identifier pattern resource according to the string parameter. This pattern, along with the identifier mode (set by the :SBUS<n>:CAN:TRIGger:PATTern:ID:MODE command), control the identifier pattern searched for in each CAN message.

If the string parameter starts with "0x", it is a hexadecimal string made up of hexadecimal and X (don't care) characters; otherwise, it is a binary string made up of 0, 1, and X (don't care) characters.

NOTE

The ID pattern resource string is always 29 bits. Only 11 of these bits are used when the :SBUS<n>:CAN:TRIGger:PATTern:ID:MODE is STANdard.

A string longer than 29 bits is truncated to 29 bits when setting the ID pattern resource.

Query Syntax

```
:SBUS<n>:CAN:TRIGger:PATTern:ID?
```

The :SBUS<n>:CAN:TRIGger:PATTern:ID? query returns the current settings of the specified CAN identifier pattern resource in the 29-bit binary string format.

Return Format

```
<string><NL> in 29-bit binary string format
```

Errors

- "-241, Hardware missing" on page 1247

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":SBUS<n>:CAN:TRIGger:PATTern:ID:MODE" on page 746
- ":SBUS<n>:CAN:TRIGger:PATTern:DATA" on page 741

:SBUS<n>:CAN:TRIGger:PATTern:ID:MODE

 (see page 1292)

Command Syntax

```
:SBUS<n>:CAN:TRIGger:PATTern:ID:MODE <value>

<value> ::= {STANdard | EXTended}
```

The :SBUS<n>:CAN:TRIGger: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 :SBUS<n>:CAN:TRIGger:PATTern:ID command.

Query Syntax

```
:SBUS<n>:CAN:TRIGger:PATTern:ID:MODE?
```

The :SBUS<n>:CAN:TRIGger:PATTern:ID:MODE? query returns the current setting of the CAN identifier mode.

Return Format

```
<value><NL>

<value> ::= {STAN | EXT}
```

Errors

- "-241, Hardware missing" on page 1247

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":SBUS<n>:MODE" on page 719
- ":SBUS<n>:CAN:TRIGger:PATTern:DATA" on page 741
- ":SBUS<n>:CAN:TRIGger:PATTern:DATA:LENGth" on page 743
- ":SBUS<n>:CAN:TRIGger:PATTern:ID" on page 745

:SBUS<n>:CAN:TRIGger:SYMBolic:MESSage

N (see page 1292)

Command Syntax

```
:SBUS<n>:CAN:TRIGger:SYMBolic:MESSage <name>

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

The :SBUS<n>:CAN:TRIGger:SYMBolic:MESSage command specifies the message to trigger on when CAN symbolic data has been loaded (recalled) into the oscilloscope and the CAN trigger mode is set to MESSage or MSIGnal.

Query Syntax

```
:SBUS<n>:CAN:TRIGger:SYMBolic:MESSage?
```

The :SBUS<n>:CAN:TRIGger:SYMBolic:MESSage? query returns the specified message.

Return Format

```
<name><NL>

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

See Also

- ":RECall:DBC[:STARt]" on page 679
- ":SBUS<n>:CAN:TRIGger" on page 737
- ":SBUS<n>:CAN:TRIGger:SYMBolic:SIGNal" on page 748
- ":SBUS<n>:CAN:TRIGger:SYMBolic:VALue" on page 749

:SBUS<n>:CAN:TRIGger:SYMBolic:SIGNal

N (see page 1292)

Command Syntax

```
:SBUS<n>:CAN:TRIGger:SYMBolic:SIGNal <name>

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

The :SBUS<n>:CAN:TRIGger:SYMBolic:SIGNal command specifies the signal to trigger on when CAN symbolic data has been loaded (recalled) into the oscilloscope and the CAN trigger mode is set to MSIGnal.

Query Syntax

```
:SBUS<n>:CAN:TRIGger:SYMBolic:SIGNal?
```

The :SBUS<n>:CAN:TRIGger:SYMBolic:SIGNal? query returns the specified signal.

Return Format

```
<name><NL>

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

See Also

- ":RECall:DBC[:STARt]" on page 679
- ":SBUS<n>:CAN:TRIGger" on page 737
- ":SBUS<n>:CAN:TRIGger:SYMBolic:MESSage" on page 747
- ":SBUS<n>:CAN:TRIGger:SYMBolic:VALue" on page 749

:SBUS<n>:CAN:TRIGger:SYMBolic:VALue

N (see page 1292)

Command Syntax

```
:SBUS<n>:CAN:TRIGger:SYMBolic:VALue <data>

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

The :SBUS<n>:CAN:TRIGger:SYMBolic:VALue command specifies the signal value to trigger on when CAN symbolic data has been loaded (recalled) into the oscilloscope and the CAN trigger mode is set to MSIGnal.

NOTE Encoded signal values are not supported in the remote interface (even though they can be used in the front panel graphical interface).

Query Syntax

```
:SBUS<n>:CAN:TRIGger:SYMBolic:VALue?
```

The :SBUS<n>:CAN:TRIGger:SYMBolic:VALue? query returns the specified signal value.

Return Format

```
<data><NL>

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

See Also

- ":RECall:DBC[:STARt]" on page 679
- ":SBUS<n>:CAN:TRIGger" on page 737
- ":SBUS<n>:CAN:TRIGger:SYMBolic:MESSage" on page 747
- ":SBUS<n>:CAN:TRIGger:SYMBolic:SIGNal" on page 748

:SBUS<n>:CAN:TRIGger:TYPE

N (see page 1292)

Command Syntax

```
:SBUS<n>:CAN:TRIGger:TYPE <type>

<type> ::= {STANdard | FD | XL}
```

The :SBUS<n>:CAN:TRIGger:TYPE command selects the CAN trigger type:

- STANdard – CAN Standard trigger, available with all CAN decode types.
- FD – CAN Flexible Data trigger, available with the FD (Flexible Data), XFASt (CAN XL Fast Mode), or XSIC (CAN XL SIC Mode) decode types.
- XL – CAN XL trigger, available with the XFASt (CAN XL Fast Mode) or XSIC (CAN XL SIC Mode) decode types.

The CAN decode type is selected with the :SBUS<n>:CAN:TYPE command.

Query Syntax

```
:SBUS<n>:CAN:TRIGger:TYPE?
```

The :SBUS<n>:CAN:TRIGger:TYPE? query returns the selected CAN trigger type.

Return Format

```
<type><NL>

<type> ::= {STAN | FD | XL}
```

See Also

- ":SBUS<n>:CAN:TYPE" on page 751
- ":SBUS<n>:CAN:SIGNal:BAUDrate" on page 732
- ":SBUS<n>:CAN:SIGNal:FDBaudrate" on page 734
- ":SBUS<n>:CAN:SIGNal:XLBaudrate" on page 735
- ":SBUS<n>:CAN:SAMPlepoint" on page 731
- ":SBUS<n>:CAN:FDSPoint" on page 730
- ":SBUS<n>:CAN:XLSPoint" on page 752

:SBUS<n>:CAN:TYPE

N (see page 1292)

Command Syntax

```
:SBUS<n>:CAN:TYPE <type>

<type> ::= {STANdard | FD | XFASt | XSIC}
```

The :SBUS<n>:CAN:TYPE command selects the CAN protocol decode type:

- STANdard – CAN Standard decode (data field length from 1 to 8 bytes).
- FD – CAN Flexible Data decode (data field length from 1 to 64 bytes).
- XFASt – CAN XL (Fast Mode) decode (arbitration mode and data mode, data field length from 1 to 2048 bytes). Fast mode is active when there is PWM encoding and decoding is happening in OSI layers.
- XSIC – CAN XL (SIC Mode) decode (slow mode, arbitration mode, same speeds as CAN-FD).

Query Syntax

```
:SBUS<n>:CAN:TYPE?
```

The :SBUS<n>:CAN:TYPE? query returns the selected CAN protocol decode type.

Return Format

```
<type><NL>

<type> ::= {STAN | FD | XFAS | XSIC}
```

See Also

- ":SBUS<n>:CAN:SIGNal:BAUDrate" on page 732
- ":SBUS<n>:CAN:SIGNal:FDBaudrate" on page 734
- ":SBUS<n>:CAN:SIGNal:XLBaudrate" on page 735
- ":SBUS<n>:CAN:SAMPlepoint" on page 731
- ":SBUS<n>:CAN:FDSPoint" on page 730
- ":SBUS<n>:CAN:XLSPoint" on page 752

:SBUS<n>:CAN:XLSPoint

N (see page 1292)

Command Syntax

```
:SBUS<n>:CAN:XLSPoint <value>

<value> ::= even numbered percentages from 30 to 90 in NR3 format.
```

The :SBUS<n>:CAN:XLSPoint 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

```
:SBUS<n>:CAN:XLSPoint?
```

The :SBUS<n>:CAN:XLSPoint? query returns the current CAN XL sample point setting.

Return Format

```
<value><NL>

<value> ::= even numbered percentages from 30 to 90 in NR3 format.
```

See Also

- ":SBUS<n>:CAN:SIGNal:XLBaudrate" on page 735
- ":SBUS<n>:CAN:TYPE" on page 751

:SBUS<n>:IIC Commands

NOTE These commands are valid when the low-speed IIC and SPI serial decode option has been licensed.

Table 97 :SBUS<n>:IIC Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:IIC:ASIZe <size> (see page 754)	:SBUS<n>:IIC:ASIZe? (see page 754)	<size> ::= {BIT7 | BIT8}
:SBUS<n>:IIC[:SOURce]:CLOCk <source> (see page 755)	:SBUS<n>:IIC[:SOURce]:CLOCk? (see page 755)	<source> ::= {CHANnel<n> | DIGital<d> } <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:SBUS<n>:IIC[:SOURce]:DATA <source> (see page 756)	:SBUS<n>:IIC[:SOURce]:DATA? (see page 756)	<source> ::= {CHANnel<n> | DIGital<d> } <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:SBUS<n>:IIC:TRIGger:PATTern:ADDRess <value> (see page 757)	:SBUS<n>:IIC:TRIGger:PATTern:ADDRess? (see page 757)	<value> ::= integer
:SBUS<n>:IIC:TRIGger:PATTern:DATA <value> (see page 758)	:SBUS<n>:IIC:TRIGger:PATTern:DATA? (see page 758)	<value> ::= integer
:SBUS<n>:IIC:TRIGger:PATTern:DATa2 <value> (see page 759)	:SBUS<n>:IIC:TRIGger:PATTern:DATa2? (see page 759)	<value> ::= integer
:SBUS<n>:IIC:TRIGger:QUALifier <value> (see page 760)	:SBUS<n>:IIC:TRIGger:QUALifier? (see page 760)	<value> ::= {EQUal | NOTequal | LESSthan | GREaterthan}
:SBUS<n>:IIC:TRIGger[:TYPE] <type> (see page 761)	:SBUS<n>:IIC:TRIGger[:TYPE]? (see page 761)	<type> ::= {STARt | STOP | RESTart | ADDRess | ANACk | DNACk | NACKnowledge | REPRom | READ7 | WRITe7 | R7Data2 | W7Data2 | WRITe10}

:SBUS<n>:IIC:ASIZe

N (see page 1292)

Command Syntax

```
:SBUS<n>:IIC:ASIZe <size>

<size> ::= {BIT7 | BIT8}
```

The :SBUS<n>: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.

Query Syntax

```
:SBUS<n>:IIC:ASIZe?
```

The :SBUS<n>:IIC:ASIZe? query returns the current IIC address width setting.

Return Format

```
<mode><NL>

<mode> ::= {BIT7 | BIT8}
```

Errors

- "-241, Hardware missing" on page 1247

See Also

- "Introduction to :SBUS<n> Commands" on page 715
- ":SBUS<n>:IIC Commands" on page 753

:SBUS<n>:IIC[:SOURce]:CLOCk

 (see page 1292)

Command Syntax

```
:SBUS<n>:IIC[:SOURce]:CLOCk <source>

<source> ::= {CHANnel<n> | DIGital<d>}

<n> ::= 1 to (# analog channels) in NR1 format

<d> ::= 0 to (# digital channels - 1) in NR1 format
```

The :SBUS<n>:IIC[:SOURce]:CLOCk command sets the source for the IIC serial clock (SCL).

Query Syntax

```
:SBUS<n>:IIC[:SOURce]:CLOCk?
```

The :SBUS<n>:IIC[:SOURce]:CLOCk? query returns the current source for the IIC serial clock.

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":SBUS<n>:IIC[:SOURce]:DATA" on page 756

:SBUS<n>:IIC[:SOURce]:DATA

N (see page 1292)

Command Syntax

```
:SBUS<n>:IIC[:SOURce]:DATA <source>

<source> ::= {CHANnel<n> | DIGital<d>}

<n> ::= 1 to (# analog channels) in NR1 format

<d> ::= 0 to (# digital channels - 1) in NR1 format
```

The :SBUS<n>:IIC[:SOURce]:DATA command sets the source for IIC serial data (SDA).

Query Syntax

```
:SBUS<n>:IIC[:SOURce]:DATA?
```

The :SBUS<n>:IIC[:SOURce]:DATA? query returns the current source for IIC serial data.

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":SBUS<n>:IIC[:SOURce]:CLOCk" on page 755

:SBUS<n>:IIC:TRIGger:PATTern:ADDRess

N (see page 1292)

Command Syntax

```
:SBUS<n>:IIC:TRIGger:PATTern:ADDRess <value>

<value> ::= integer
```

The :SBUS<n>:IIC:TRIGger:PATTern:ADDRess command sets the address for IIC data.The address can range from 0 to 127 (7-bit) or 1023 (10-bit). Use the don't care address (-1) to ignore the address value.

Query Syntax

```
:SBUS<n>:IIC:TRIGger:PATTern:ADDRess?
```

The :SBUS<n>:IIC:TRIGger:PATTern:ADDRess? query returns the current address for IIC data.

Return Format

```
<value><NL>

<value> ::= integer
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":SBUS<n>:IIC:TRIGger:PATTern:DATA" on page 758
- ":SBUS<n>:IIC:TRIGger:PATTern:DATa2" on page 759
- ":SBUS<n>:IIC:TRIGger[:TYPE]" on page 761

:SBUS<n>:IIC:TRIGger:PATTern:DATA

N (see page 1292)

Command Syntax

```
:SBUS<n>:IIC:TRIGger:PATTern:DATA <value>

<value> ::= integer
```

The :SBUS<n>:IIC:TRIGger:PATTern:DATA command sets IIC data. The data value can range from 0 to 255. Use the don't care data pattern (-1) to ignore the data value.

Query Syntax

```
:SBUS<n>:IIC:TRIGger:PATTern:DATA?
```

The :SBUS<n>:IIC:TRIGger:PATTern:DATA? query returns the current pattern for IIC data.

Return Format

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

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":SBUS<n>:IIC:TRIGger:PATTern:ADDRess" on page 757
- ":SBUS<n>:IIC:TRIGger:PATTern:DATa2" on page 759
- ":SBUS<n>:IIC:TRIGger[:TYPE]" on page 761
- "Commands Not Supported in Multiple Program Message Units" on page 1297

:SBUS<n>:IIC:TRIGger:PATTern:DATa2

N (see page 1292)

Command Syntax

```
:SBUS<n>:IIC:TRIGger:PATTern:DATa2 <value>

<value> ::= integer
```

The :SBUS<n>:IIC:TRIGger:PATTern:DATa2 command sets IIC data 2. The data value can range from 0 to 255. Use the don't care data pattern (-1) to ignore the data value.

Query Syntax

```
:SBUS<n>:IIC:TRIGger:PATTern:DATa2?
```

The :SBUS<n>:IIC:TRIGger:PATTern:DATa2? query returns the current pattern for IIC data 2.

Return Format

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

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":SBUS<n>:IIC:TRIGger:PATTern:ADDRess" on page 757
- ":SBUS<n>:IIC:TRIGger:PATTern:DATA" on page 758
- ":SBUS<n>:IIC:TRIGger[:TYPE]" on page 761
- "Commands Not Supported in Multiple Program Message Units" on page 1297

:SBUS<n>:IIC:TRIGger:QUALifier

(see page 1292)

Command Syntax

```
:SBUS<n>:IIC:TRIGger:QUALifier <value>

<value> ::= {EQUal | NOTequal | LESSthan | GREaterthan}
```

The :SBUS<n>:IIC:TRIGger:QUALifier command sets the IIC data qualifier when TRIGger:IIC:TRIGger[:TYPE] is set to REPRom.

Query Syntax

```
:SBUS<n>:IIC:TRIGger:QUALifier?
```

The :SBUS<n>: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" on page 1077
- ":TRIGger:MODE" on page 1091
- ":SBUS<n>:IIC:TRIGger[:TYPE]" on page 761

:SBUS<n>:IIC:TRIGger[:TYPE]

N (see page 1292)

Command Syntax

```
:SBUS<n>:IIC:TRIGger[:TYPE] <value>

<value> ::= {STARt | STOP | RESTart | ADDRess | ANACk | DNACk
   | NACKnowledge | REPRom | READ7 | WRITe7 | R7Data2 | W7Data2
   | WRITe10}
```

The :SBUS<n>:IIC:TRIGger[:TYPE] command sets the IIC trigger type:

- STARt – Start condition.
- STOP – Stop condition.
- RESTart – Another start condition occurs before a stop condition.
- ADDRess – Triggers on the selected address. The R/W bit is ignored.
- ANACk – Address with no acknowledge.
- DNACk – Write data with no acknowledge.
- NACKnowledge – Missing acknowledge.
- REPRom – EEPROM data read.
- READ7 – 7-bit address frame containing (Start:Address7:Read:Ack:Data). The value READ is also accepted for READ7.
- WRITe7 – 7-bit address frame containing (Start:Address7:Write:Ack:Data). The value WRITe is also accepted for WRITe7.
- R7Data2 – 7-bit address frame containing (Start:Address7:Read:Ack:Data:Ack:Data2).
- 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).

NOTE **The short form of READ7 (READ7), REPRom (REPR), WRITe7 (WRIT7), and WRITe10 (WRIT10) do not follow the defined Long Form to Short Form Truncation Rules (see page 1294).**

Query Syntax

```
:SBUS<n>:IIC:TRIGger[:TYPE]?
```

The :SBUS<n>:IIC:TRIGger[:TYPE]? query returns the current IIC trigger type value.

Return Format

```
<value><NL>

<value> ::= {STAR | STOP | REST | ADDR | ANAC | DNAC | NACK | REPR
   | READ7 | WRIT7 | R7D2 | W7D2 | WRIT10}
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:MODE" on page 1091

- ":SBUS<n>:IIC:TRIGger:PATTern:ADDRess" on page 757
- ":SBUS<n>:IIC:TRIGger:PATTern:DATA" on page 758
- ":SBUS<n>:IIC:TRIGger:PATTern:DATa2" on page 759
- ":SBUS<n>:IIC:TRIGger:QUALifier" on page 760
- "Long Form to Short Form Truncation Rules" on page 1294

:SBUS<n>:LIN Commands

NOTE These commands are valid when the CAN and LIN serial decode license has been enabled.

Table 98 :SBUS<n>:LIN Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:LIN:DISPlay <type> (see page 765)	:SBUS<n>:LIN:DISPlay? (see page 765)	<type> ::= {HEXadecimal | SYMBolic}
:SBUS<n>:LIN:PARity {{0 | OFF} | {1 | ON}} (see page 766)	:SBUS<n>:LIN:PARity? (see page 766)	{0 | 1}
:SBUS<n>:LIN:SAMPlepoint <value> (see page 767)	:SBUS<n>:LIN:SAMPlepoint? (see page 767)	<value> ::= {60 | 62.5 | 68 | 70 | 75 | 80 | 87.5} in NR3 format
:SBUS<n>:LIN:SIGNal:BAUDrate <baudrate> (see page 768)	:SBUS<n>:LIN:SIGNal:BAUDrate? (see page 768)	<baudrate> ::= integer from 2400 to 625000 in 100 b/s increments
:SBUS<n>:LIN:SOURce <source> (see page 769)	:SBUS<n>:LIN:SOURce? (see page 769)	<source> ::= {CHANnel<n> | DIGital<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:SBUS<n>:LIN:STANdard <std> (see page 770)	:SBUS<n>:LIN:STANdard? (see page 770)	<std> ::= {LIN13 | LIN13NLC | LIN20}
:SBUS<n>:LIN:SYNCbreak <value> (see page 771)	:SBUS<n>:LIN:SYNCbreak? (see page 771)	<value> ::= integer = {11 | 12 | 13}
:SBUS<n>:LIN:TRIGger <condition> (see page 772)	:SBUS<n>:LIN:TRIGger? (see page 772)	<condition> ::= {SYNCbreak | ID | DATA}

Table 98 :SBUS<n>:LIN Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:LIN:TRIGger:ID <value> (see page 773)	:SBUS<n>:LIN:TRIGger:ID? (see page 773)	<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
:SBUS<n>:LIN:TRIGger:PATTern:DATA <string> (see page 774)	:SBUS<n>:LIN:TRIGger:PATTern:DATA? (see page 774)	<string> ::= "n" where n ::= 32-bit integer in unsigned decimal when <base> = DECimal <string> ::= "nn...n" where n ::= {0 | 1 | X | $} when <base> = BINary <string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $} when <base> = HEX
:SBUS<n>:LIN:TRIGger:PATTern:DATA:LENGth <length> (see page 776)	:SBUS<n>:LIN:TRIGger:PATTern:DATA:LENGth? (see page 776)	<length> ::= integer from 1 to 8 in NR1 format
:SBUS<n>:LIN:TRIGger:PATTern:FORMat <base> (see page 777)	:SBUS<n>:LIN:TRIGger:PATTern:FORMat? (see page 777)	<base> ::= {BINary | HEX | DECimal}
:SBUS<n>:LIN:TRIGger:SYMBolic:FRAMe <name> (see page 778)	:SBUS<n>:LIN:TRIGger:SYMBolic:FRAMe? (see page 778)	<name> ::= quoted ASCII string
:SBUS<n>:LIN:TRIGger:SYMBolic:SIGNal <name> (see page 779)	:SBUS<n>:LIN:TRIGger:SYMBolic:SIGNal? (see page 779)	<name> ::= quoted ASCII string
:SBUS<n>:LIN:TRIGger:SYMBolic:VALue <data> (see page 780)	:SBUS<n>:LIN:TRIGger:SYMBolic:VALue? (see page 780)	<data> ::= value in NR3 format

:SBUS<n>:LIN:DISPlay

N (see page 1292)

Command Syntax

```
:SBUS<n>:LIN:DISPlay <type>

<type> ::= {HEXadecimal | SYMBolic}
```

The :SBUS<n>:LIN:DISPlay command specifies, when LIN symbolic data is loaded into the oscilloscope, whether symbolic values (from the LDF file) or hexadecimal values are displayed in the decode waveform and the Lister window.

Query Syntax

```
:SBUS<n>:LIN:DISPlay?
```

The :SBUS<n>:LIN:DISPlay? query returns the LIN decode display type.

Return Format

```
<type><NL>

<type> ::= {HEX | SYMB}
```

See Also

- ":RECall:LDF[:STARt]" on page 681

:SBUS<n>:LIN:PARity

N (see page 1292)

Command Syntax

```
:SBUS<n>:LIN:PARity <display>

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

The :SBUS<n>: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.

Query Syntax

```
:SBUS<n>:LIN:PARity?
```

The :SBUS<n>:LIN:PARity? query returns the current LIN parity bits display setting of the serial decode bus.

Return Format

```
<display><NL>

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

Errors

- "-241, Hardware missing" on page 1247

See Also

- "Introduction to :SBUS<n> Commands" on page 715
- ":SBUS<n>:LIN Commands" on page 763

:SBUS<n>:LIN:SAMPlepoint

(see page 1292)

Command Syntax

```
:SBUS<n>:LIN:SAMPlepoint <value>

<value><NL>

<value> ::= {60 | 62.5 | 68 | 70 | 75 | 80 | 87.5} in NR3 format
```

The :SBUS<n>: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

```
:SBUS<n>:LIN:SAMPlepoint?
```

The :SBUS<n>: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" on page 1077
- ":TRIGger:MODE" on page 1091
- ":SBUS<n>:LIN:TRIGger" on page 772

:SBUS<n>:LIN:SIGNal:BAUDrate

 (see page 1292)

Command Syntax

```
:SBUS<n>:LIN:SIGNal:BAUDrate <baudrate>

<baudrate> ::= integer from 2400 to 625000 in 100 b/s increments
```

The :SBUS<n>:LIN:SIGNal:BAUDrate command sets the standard baud rate of the LIN signal from 2400 b/s to 625 kb/s in 100 b/s increments. If you enter a baud rate that is not divisible by 100 b/s, the baud rate is set to the nearest baud rate divisible by 100 b/s.

Query Syntax

```
:SBUS<n>:LIN:SIGNal:BAUDrate?
```

The :SBUS<n>:LIN:SIGNal:BAUDrate? query returns the current LIN baud rate setting.

Return Format

```
<baudrate><NL>

<baudrate> ::= integer from 2400 to 625000 in 100 b/s increments
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:MODE" on page 1091
- ":SBUS<n>:LIN:TRIGger" on page 772
- ":SBUS<n>:LIN:SOURce" on page 769

:SBUS<n>:LIN:SOURce

N (see page 1292)

Command Syntax

```
:SBUS<n>:LIN:SOURce <source>

<source> ::= {CHANnel<n> | DIGital<d>}

<n> ::= 1 to (# analog channels) in NR1 format

<d> ::= 0 to (# digital channels - 1) in NR1 format
```

The :SBUS<n>:LIN:SOURce command sets the source for the LIN signal.

Query Syntax

```
:SBUS<n>:LIN:SOURce?
```

The :SBUS<n>:LIN:SOURce? query returns the current source for the LIN signal.

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:MODE" on page 1091
- ":SBUS<n>:LIN:TRIGger" on page 772

:SBUS<n>:LIN:STANdard

N (see page 1292)

Command Syntax

```
:SBUS<n>:LIN:STANdard <std>

<std> ::= {LIN13 | LIN13NLC | LIN20}
```

The :SBUS<n>:LIN:STANdard command sets the LIN standard in effect for triggering and decoding:

- LIN13 – LIN 1.3.
- LIN13NLC – LIN 1.3 (no length control). Select this for systems where length control is not used and all nodes have knowledge of the data packet size. In LIN 1.3, the ID may or may not be used to indicate the number of bytes. (In LIN 2.X, there is no length control.)
- LIN20 – LIN 2.X.

For LIN 1.2 signals, use the LIN 1.3 setting. The LIN 1.3 setting assumes the signal follows the "Table of Valid ID Values" as shown in section A.2 of the LIN Specification dated December 12, 2002. If your signal does not comply with the table, use the LIN 2.X setting.

Query Syntax

```
:SBUS<n>:LIN:STANdard?
```

The :SBUS<n>:LIN:STANdard? query returns the current LIN standard setting.

Return Format

```
<std><NL>

<std> ::= {LIN13 | LIN13NLC | LIN20}
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:MODE" on page 1091
- ":SBUS<n>:LIN:SOURce" on page 769

:SBUS<n>:LIN:SYNCbreak

 (see page 1292)

Command Syntax

```
:SBUS<n>:LIN:SYNCbreak <value>

<value> ::= integer = {11 | 12 | 13}
```

The :SBUS<n>: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

```
:SBUS<n>:LIN:SYNCbreak?
```

The :SBUS<n>:LIN:SYNCbreak? query returns the current LIN sync break setting.

Return Format

```
<value><NL>

<value> ::= {11 | 12 | 13}
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:MODE" on page 1091
- ":SBUS<n>:LIN:SOURce" on page 769

:SBUS<n>:LIN:TRIGger

N (see page 1292)

Command Syntax

```
:SBUS<n>:LIN:TRIGger <condition>

<condition> ::= {SYNCbreak | ID | DATA | PARityerror | CSUMerror
                 | FRAMe | FSIGnal}
```

The :SBUS<n>:LIN:TRIGger command sets the LIN trigger condition to be:

- SYNCbreak — Sync Break.
- ID — Frame ID.

  Use the :SBUS<n>:LIN:TRIGger:ID command to specify the frame ID.
- DATA — Frame ID and Data.

  Use the :SBUS<n>:LIN:TRIGger:ID command to specify the frame ID.

  Use the :SBUS<n>:LIN:TRIGger:PATTern:DATA:LENGth and :SBUS<n>:LIN:TRIGger:PATTern:DATA commands to specify the data string length and value.
- PARityerror — parity errors.
- CSUMerror — checksum errors.
- FRAMe — Triggers on a symbolic frame.
- FSIGnal — Triggers on a symbolic frame and a signal value.

Query Syntax

```
:SBUS<n>:LIN:TRIGger?
```

The :SBUS<n>:LIN:TRIGger? query returns the current LIN trigger value.

Return Format

```
<condition><NL>

<condition> ::= {SYNC | ID | DATA | PAR | CSUM | FRAM | FSIG}
```

Errors

- "-241, Hardware missing" on page 1247

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:MODE" on page 1091
- ":SBUS<n>:LIN:TRIGger:ID" on page 773
- ":SBUS<n>:LIN:TRIGger:PATTern:DATA:LENGth" on page 776
- ":SBUS<n>:LIN:TRIGger:PATTern:DATA" on page 774
- ":SBUS<n>:LIN:SOURce" on page 769
- ":RECall:LDF[:STARt]" on page 681
- ":SBUS<n>:LIN:TRIGger:SYMBolic:FRAMe" on page 778
- ":SBUS<n>:LIN:TRIGger:SYMBolic:SIGNal" on page 779
- ":SBUS<n>:LIN:TRIGger:SYMBolic:VALue" on page 780

:SBUS<n>:LIN:TRIGger:ID

(see page 1292)

Command Syntax

```
:SBUS<n>:LIN:TRIGger: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 :SBUS<n>:LIN:TRIGger:ID command defines the LIN identifier searched for in each CAN message when the LIN trigger mode is set to frame ID.

Setting the ID to a value of "-1" results in "0xXX" which is equivalent to all IDs.

Query Syntax

```
:SBUS<n>:LIN:TRIGger:ID?
```

The :SBUS<n>:LIN:TRIGger:ID? query returns the current LIN identifier setting.

Return Format

```
<value><NL>

<value> ::= integer in decimal
```

Errors

- "-241, Hardware missing" on page 1247

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:MODE" on page 1091
- ":SBUS<n>:LIN:TRIGger" on page 772
- ":SBUS<n>:LIN:SOURce" on page 769

:SBUS<n>:LIN:TRIGger:PATTern:DATA

N (see page 1292)

Command Syntax

```
:SBUS<n>:LIN:TRIGger:PATTern:DATA <string>

<string> ::= "n" where n ::= 32-bit integer in unsigned decimal when
             <base> = DECimal

<string> ::= "nn...n" where n ::= {0 | 1 | X | $} when
             <base> = BINary

<string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $} when
             <base> = HEX
```

NOTE <base> is specified with the :SBUS<n>:LIN:TRIGger:PATTern:FORMat command. The default <base> is BINary.

The :SBUS<n>:LIN:TRIGger:PATTern:DATA command specifies the LIN trigger data pattern searched for in each LIN data field.

Set a <string> bit to "0" or "1" to set the corresponding bit in the data pattern to low or high, respectively.

Set a <string> bit to "X" to ignore (mask off) that bit in the data pattern.

Use the "$" character to indicate that the value of the corresponding bit will not be changed (the existing bit value is used).

When <base> = DECimal, the "X" and "$" characters cannot be entered. When queried, the "$" character is returned when any bits in the pattern have the value of "X" and <base> = DECimal. When any bits in a given nibble have the value of "X" and <base> = HEX, the "$" character is returned for the corresponding nibble.

NOTE The length of the trigger data value is determined by the :SBUS<n>:LIN:TRIGger:PATTern:DATA:LENGth command.

NOTE If more bits are sent for <string> than the specified trigger pattern data length, the most significant bits will be truncated. If the data length size is changed after the <string> is programmed, the added or deleted bits will be added to or deleted from the least significant bits.

Query Syntax

```
:SBUS<n>:LIN:TRIGger:PATTern:DATA?
```

The :SBUS<n>:LIN:TRIGger:PATTern:DATA? query returns the currently specified LIN trigger data pattern.

Return Format

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

See Also
- "Introduction to :TRIGger Commands" on page 1077
- ":SBUS<n>:LIN:TRIGger:PATTern:FORMat" on page 777
- ":SBUS<n>:LIN:TRIGger" on page 772
- ":SBUS<n>:LIN:TRIGger:PATTern:DATA:LENGth" on page 776

:SBUS<n>:LIN:TRIGger:PATTern:DATA:LENGth

N (see page 1292)

Command Syntax

```
:SBUS<n>:LIN:TRIGger:PATTern:DATA:LENGth <length>

<length> ::= integer from 1 to 8 in NR1 format
```

The :SBUS<n>:LIN:TRIGger:PATTern:DATA:LENGth command sets the number of 8-bit bytes in the LIN data string. The number of bytes in the string can be anywhere from 1 bytes to 8 bytes (64 bits). The value for these bytes is set by the :SBUS<n>:LIN:TRIGger:PATTern:DATA command.

Query Syntax

```
:SBUS<n>:LIN:TRIGger:PATTern:DATA:LENGth?
```

The :SBUS<n>:LIN:TRIGger:PATTern:DATA:LENGth? query returns the current LIN data pattern length setting.

Return Format

```
<count><NL>

<count> ::= integer from 1 to 8 in NR1 format
```

Errors

- "-241, Hardware missing" on page 1247

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":SBUS<n>:LIN:TRIGger:PATTern:DATA" on page 774
- ":SBUS<n>:LIN:SOURce" on page 769

:SBUS<n>:LIN:TRIGger:PATTern:FORMat

N (see page 1292)

Command Syntax

```
:SBUS<n>:LIN:TRIGger:PATTern:FORMat <base>

<base> ::= {BINary | HEX | DECimal}
```

The :SBUS<n>:LIN:TRIGger:PATTern:FORMat command sets the entry (and query) number base used by the :SBUS<n>:LIN:TRIGger:PATTern:DATA command. The default <base> is BINary.

Query Syntax

```
:SBUS<n>:LIN:TRIGger:PATTern:FORMat?
```

The :SBUS<n>:LIN:TRIGger:PATTern:FORMat? query returns the currently set number base for LIN pattern data.

Return Format

```
<base><NL>

<base> ::= {BIN | HEX | DEC}
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":SBUS<n>:LIN:TRIGger:PATTern:DATA" on page 774
- ":SBUS<n>:LIN:TRIGger:PATTern:DATA:LENGth" on page 776

:SBUS<n>:LIN:TRIGger:SYMBolic:FRAMe

(see page 1292)

Command Syntax

```
:SBUS<n>:LIN:TRIGger:SYMBolic:FRAMe <name>

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

The :SBUS<n>:LIN:TRIGger:SYMBolic:FRAMe command specifies the message to trigger on when LIN symbolic data has been loaded (recalled) into the oscilloscope and the LIN trigger mode is set to FRAMe or FSIGnal.

Query Syntax

```
:SBUS<n>:LIN:TRIGger:SYMBolic:FRAMe?
```

The :SBUS<n>:LIN:TRIGger:SYMBolic:FRAMe? query returns the specified message.

Return Format

```
<name><NL>

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

See Also

- ":RECall:LDF[:STARt]" on page 681
- ":SBUS<n>:LIN:TRIGger" on page 772
- ":SBUS<n>:LIN:TRIGger:SYMBolic:SIGNal" on page 779
- ":SBUS<n>:LIN:TRIGger:SYMBolic:VALue" on page 780

:SBUS<n>:LIN:TRIGger:SYMBolic:SIGNal

N (see page 1292)

Command Syntax

```
:SBUS<n>:LIN:TRIGger:SYMBolic:SIGNal <name>

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

The :SBUS<n>:LIN:TRIGger:SYMBolic:SIGNal command specifies the signal to trigger on when LIN symbolic data has been loaded (recalled) into the oscilloscope and the LIN trigger mode is set to FSIGnal.

Query Syntax

```
:SBUS<n>:LIN:TRIGger:SYMBolic:SIGNal?
```

The :SBUS<n>:LIN:TRIGger:SYMBolic:SIGNal? query returns the specified signal.

Return Format

```
<name><NL>

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

See Also

- ":RECall:LDF[:STARt]" on page 681
- ":SBUS<n>:LIN:TRIGger" on page 772
- ":SBUS<n>:LIN:TRIGger:SYMBolic:FRAMe" on page 778
- ":SBUS<n>:LIN:TRIGger:SYMBolic:VALue" on page 780

:SBUS<n>:LIN:TRIGger:SYMBolic:VALue

N (see page 1292)

Command Syntax

```
:SBUS<n>:LIN:TRIGger:SYMBolic:VALue <data>

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

The :SBUS<n>:LIN:TRIGger:SYMBolic:VALue command specifies the signal value to trigger on when LIN symbolic data has been loaded (recalled) into the oscilloscope and the LIN trigger mode is set to FSIGnal.

NOTE Encoded signal values are not supported in the remote interface (even though they can be used in the front panel graphical interface).

Query Syntax

```
:SBUS<n>:LIN:TRIGger:SYMBolic:VALue?
```

The :SBUS<n>:LIN:TRIGger:SYMBolic:VALue? query returns the specified signal value.

Return Format

```
<data><NL>

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

See Also

- ":RECall:LDF[:STARt]" on page 681
- ":SBUS<n>:LIN:TRIGger" on page 772
- ":SBUS<n>:LIN:TRIGger:SYMBolic:FRAMe" on page 778
- ":SBUS<n>:LIN:TRIGger:SYMBolic:SIGNal" on page 779

:SBUS<n>:SPI Commands

NOTE These commands are only valid when the low-speed IIC and SPI serial decode option has been licensed.

Table 99 :SBUS<n>:SPI Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:SPI:BITorder <order> (see page 783)	:SBUS<n>:SPI:BITorder? (see page 783)	<order> ::= {LSBFirst | MSBFirst}
:SBUS<n>:SPI:CLOCk:SLOPe <slope> (see page 784)	:SBUS<n>:SPI:CLOCk:SLOPe? (see page 784)	<slope> ::= {NEGative | POSitive}
:SBUS<n>:SPI:CLOCk:TIMeout <time_value> (see page 785)	:SBUS<n>:SPI:CLOCk:TIMeout? (see page 785)	<time_value> ::= time in seconds in NR3 format
:SBUS<n>:SPI:DELay <value> (see page 786)	:SBUS<n>:SPI:DELay? (see page 786)	<value> ::= {OFF | 2-63}
:SBUS<n>:SPI:FRAMing <value> (see page 787)	:SBUS<n>:SPI:FRAMing? (see page 787)	<value> ::= {CHIPselect | {NCHipselect | NOTC} | TIMeout}
:SBUS<n>:SPI:SOURce:CLOCk <source> (see page 788)	:SBUS<n>:SPI:SOURce:CLOCk? (see page 788)	<value> ::= {CHANnel<n> | DIGital<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:SBUS<n>:SPI:SOURce:FRAMe <source> (see page 789)	:SBUS<n>:SPI:SOURce:FRAMe? (see page 789)	<value> ::= {CHANnel<n> | DIGital<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:SBUS<n>:SPI:SOURce:MISO <source> (see page 790)	:SBUS<n>:SPI:SOURce:MISO? (see page 790)	<value> ::= {CHANnel<n> | DIGital<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format

Table 99 :SBUS<n>:SPI Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:SPI:SOURce:MOSI <source> (see page 791)	:SBUS<n>:SPI:SOURce:MOSI? (see page 791)	<value> ::= {CHANnel<n> | DIGital<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:SBUS<n>:SPI:TRIGger:PATTern:MISO:DATA <string> (see page 792)	:SBUS<n>:SPI:TRIGger:PATTern:MISO:DATA? (see page 792)	<string> ::= "nn...n" where n ::= {0 | 1 | X | $} <string ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $}
:SBUS<n>:SPI:TRIGger:PATTern:MISO:WIDTh <width> (see page 793)	:SBUS<n>:SPI:TRIGger:PATTern:MISO:WIDTh? (see page 793)	<width> ::= integer from 4 to 64 in NR1 format
:SBUS<n>:SPI:TRIGger:PATTern:MOSI:DATA <string> (see page 794)	:SBUS<n>:SPI:TRIGger:PATTern:MOSI:DATA? (see page 794)	<string> ::= "nn...n" where n ::= {0 | 1 | X | $} <string ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $}
:SBUS<n>:SPI:TRIGger:PATTern:MOSI:WIDTh <width> (see page 795)	:SBUS<n>:SPI:TRIGger:PATTern:MOSI:WIDTh? (see page 795)	<width> ::= integer from 4 to 64 in NR1 format
:SBUS<n>:SPI:TRIGger:TYPE <value> (see page 796)	:SBUS<n>:SPI:TRIGger:TYPE? (see page 796)	<value> ::= {MOSI | MISO}
:SBUS<n>:SPI:WIDTh <word_width> (see page 797)	:SBUS<n>:SPI:WIDTh? (see page 797)	<word_width> ::= integer 4-16 in NR1 format

:SBUS<n>:SPI:BITorder

(see page 1292)

Command Syntax

```
:SBUS<n>:SPI:BITorder <order>

<order> ::= {LSBFirst | MSBFirst}
```

The :SBUS<n>:SPI:BITorder command selects the bit order, most significant bit first (MSB) or least significant bit first (LSB), used when displaying data in the serial decode waveform and in the Lister.

Query Syntax

```
:SBUS<n>:SPI:BITorder?
```

The :SBUS<n>:SPI:BITorder? query returns the current SPI decode bit order.

Return Format

```
<order><NL>

<order> ::= {LSBF | MSBF}
```

Errors

- "-241, Hardware missing" on page 1247

See Also

- "Introduction to :SBUS<n> Commands" on page 715
- ":SBUS<n>:MODE" on page 719
- ":SBUS<n>:SPI Commands" on page 781

:SBUS<n>:SPI:CLOCk:SLOPe

 (see page 1292)

Command Syntax

```
:SBUS<n>:SPI:CLOCk:SLOPe <slope>

<slope> ::= {NEGative | POSitive}
```

The :SBUS<n>: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

```
:SBUS<n>:SPI:CLOCk:SLOPe?
```

The :SBUS<n>:SPI:CLOCk:SLOPe? query returns the current SPI clock source slope.

Return Format

```
<slope><NL>

<slope> ::= {NEG | POS}
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":SBUS<n>:SPI:CLOCk:TIMeout" on page 785
- ":SBUS<n>:SPI:SOURce:CLOCk" on page 788

:SBUS<n>:SPI:CLOCk:TIMeout

N (see page 1292)

Command Syntax

```
:SBUS<n>:SPI:CLOCk:TIMeout <time_value>

<time_value> ::= time in seconds in NR3 format
```

The :SBUS<n>:SPI:CLOCk:TIMeout command sets the SPI signal clock timeout resource in seconds from 100 ns to 10 s when the :SBUS<n>:SPI:FRAMing command is set to TIMeout. The timer is used to frame a signal by a clock timeout.

Query Syntax

```
:SBUS<n>:SPI:CLOCk:TIMeout?
```

The :SBUS<n>:SPI:CLOCk:TIMeout? query returns current SPI clock timeout setting.

Return Format

```
<time value><NL>

<time_value> ::= time in seconds in NR3 format
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":SBUS<n>:SPI:CLOCk:SLOPe" on page 784
- ":SBUS<n>:SPI:SOURce:CLOCk" on page 788
- ":SBUS<n>:SPI:FRAMing" on page 787

:SBUS<n>:SPI:DELay

N (see page 1292)

Command Syntax

```
:SBUS<n>:SPI:DELay <value>

<value> ::= {OFF | 2-63}
```

The :SBUS<n>:SPI:DELay command specifies the number of bits to ignore (delay) before decoding the MISO stream.

Query Syntax

```
:SBUS<n>:SPI:DELay?
```

The :SBUS<n>:SPI:DELay? query returns the specified number of delay bits or "OFF" if there are none.

Return Format

```
<value><NL>

<value> ::= {OFF | 2-63}
```

See Also

- ":SBUS<n>:SPI:SOURce:MISO" on page 790

:SBUS<n>:SPI:FRAMing

 (see page 1292)

Command Syntax

```
:SBUS<n>:SPI:FRAMing <value>

<value> ::= {CHIPselect | {NCHipselect | NOTC} | TIMeout}
```

The :SBUS<n>:SPI:FRAMing command sets the SPI trigger framing value. If TIMeout is selected, the timeout value is set by the :SBUS<n>:SPI:CLOCk:TIMeout command.

NOTE The NOTC value is deprecated. It is the same as NCHipselect.

Query Syntax

```
:SBUS<n>:SPI:FRAMing?
```

The :SBUS<n>:SPI:FRAMing? query returns the current SPI framing value.

Return Format

```
<value><NL>

<value> ::= {CHIP | NCH | TIM}
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:MODE" on page 1091
- ":SBUS<n>:SPI:CLOCk:TIMeout" on page 785
- ":SBUS<n>:SPI:SOURce:FRAMe" on page 789

:SBUS<n>:SPI:SOURce:CLOCk

N (see page 1292)

Command Syntax

```
:SBUS<n>:SPI:SOURce:CLOCk <source>

<source> ::= {CHANnel<n> | DIGital<d>}

<n> ::= 1 to (# analog channels) in NR1 format

<d> ::= 0 to (# digital channels - 1) in NR1 format
```

The :SBUS<n>:SPI:SOURce:CLOCk command sets the source for the SPI serial clock.

Query Syntax

```
:SBUS<n>:SPI:SOURce:CLOCk?
```

The :SBUS<n>:SPI:SOURce:CLOCk? query returns the current source for the SPI serial clock.

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":SBUS<n>:SPI:CLOCk:SLOPe" on page 784
- ":SBUS<n>:SPI:CLOCk:TIMeout" on page 785
- ":SBUS<n>:SPI:SOURce:FRAMe" on page 789
- ":SBUS<n>:SPI:SOURce:MOSI" on page 791
- ":SBUS<n>:SPI:SOURce:MISO" on page 790

:SBUS<n>:SPI:SOURce:FRAMe

N (see page 1292)

Command Syntax

```
:SBUS<n>:SPI:SOURce:FRAMe <source>

<source> ::= {CHANnel<n> | DIGital<d>}

<n> ::= 1 to (# analog channels) in NR1 format

<d> ::= 0 to (# digital channels - 1) in NR1 format
```

The :SBUS<n>:SPI:SOURce:FRAMe command sets the frame source when :SBUS<n>:SPI:FRAMing is set to CHIPselect or NOTChipselect.

Query Syntax

```
:SBUS<n>:SPI:SOURce:FRAMe?
```

The :SBUS<n>:SPI:SOURce:FRAMe? query returns the current frame source for the SPI serial frame.

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":SBUS<n>:SPI:SOURce:CLOCk" on page 788
- ":SBUS<n>:SPI:SOURce:MOSI" on page 791
- ":SBUS<n>:SPI:SOURce:MISO" on page 790
- ":SBUS<n>:SPI:FRAMing" on page 787

:SBUS<n>:SPI:SOURce:MISO

(see page 1292)

Command Syntax

```
:SBUS<n>:SPI:SOURce:MISO <source>

<source> ::= {CHANnel<n> | DIGital<d>}

<n> ::= 1 to (# analog channels) in NR1 format

<d> ::= 0 to (# digital channels - 1) in NR1 format
```

The :SBUS<n>:SPI:SOURce:MISO command sets the source for the SPI serial MISO data.

Query Syntax

```
:SBUS<n>:SPI:SOURce:MISO?
```

The :SBUS<n>:SPI:SOURce:MISO? query returns the current source for the SPI serial MISO data.

Return Format

```
<source><NL>
```

See Also

- ":SBUS<n>:SPI:DELay" on page 786
- "Introduction to :TRIGger Commands" on page 1077
- ":SBUS<n>:SPI:SOURce:MOSI" on page 791
- ":SBUS<n>:SPI:SOURce:CLOCk" on page 788
- ":SBUS<n>:SPI:SOURce:FRAMe" on page 789
- ":SBUS<n>:SPI:TRIGger:PATTern:MISO:DATA" on page 792
- ":SBUS<n>:SPI:TRIGger:PATTern:MOSI:DATA" on page 794
- ":SBUS<n>:SPI:TRIGger:PATTern:MISO:WIDTh" on page 793
- ":SBUS<n>:SPI:TRIGger:PATTern:MOSI:WIDTh" on page 795

:SBUS<n>:SPI:SOURce:MOSI

N (see page 1292)

Command Syntax

```
:SBUS<n>:SPI:SOURce:MOSI <source>

<source> ::= {CHANnel<n> | DIGital<d>}

<n> ::= 1 to (# analog channels) in NR1 format

<d> ::= 0 to (# digital channels - 1) in NR1 format
```

The :SBUS<n>:SPI:SOURce:MOSI command sets the source for the SPI serial MOSI data.

Query Syntax

```
:SBUS<n>:SPI:SOURce:MOSI?
```

The :SBUS<n>:SPI:SOURce:MOSI? query returns the current source for the SPI serial MOSI data.

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":SBUS<n>:SPI:SOURce:MISO" on page 790
- ":SBUS<n>:SPI:SOURce:CLOCk" on page 788
- ":SBUS<n>:SPI:SOURce:FRAMe" on page 789
- ":SBUS<n>:SPI:TRIGger:PATTern:MISO:DATA" on page 792
- ":SBUS<n>:SPI:TRIGger:PATTern:MOSI:DATA" on page 794
- ":SBUS<n>:SPI:TRIGger:PATTern:MISO:WIDTh" on page 793
- ":SBUS<n>:SPI:TRIGger:PATTern:MOSI:WIDTh" on page 795

:SBUS<n>:SPI:TRIGger:PATTern:MISO:DATA

N (see page 1292)

Command Syntax

```
:SBUS<n>:SPI:TRIGger:PATTern:MISO:DATA <string>

<string> ::= "nn...n" where n ::= {0 | 1 | X | $}

<string ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $}
```

The :SBUS<n>:SPI:TRIGger:PATTern:MISO:DATA command defines the SPI data pattern resource according to the string parameter. This pattern, along with the data width, control the data pattern searched for in the data stream.

If the string parameter starts with "0x", it is a hexadecimal string made up of hexadecimal and X (don't care) characters; otherwise, it is a binary string made up of 0, 1, and X (don't care) characters.

NOTE **The :SBUS<n>:SPI:TRIGger:PATTern:MISO:WIDTh should be set before :SBUS<n>:SPI:TRIGger:PATTern:MISO:DATA.**

Query Syntax

```
:SBUS<n>:SPI:TRIGger:PATTern:MISO:DATA?
```

The :SBUS<n>:SPI:TRIGger:PATTern:MISO:DATA? query returns the current settings of the specified SPI data pattern resource in the binary string format.

Return Format

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

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":SBUS<n>:SPI:TRIGger:PATTern:MISO:WIDTh" on page 793
- ":SBUS<n>:SPI:SOURce:MISO" on page 790

:SBUS<n>:SPI:TRIGger:PATTern:MISO:WIDTh

N (see page 1292)

Command Syntax

```
:SBUS<n>:SPI:TRIGger:PATTern:MISO:WIDTh <width>

<width> ::= integer from 4 to 64 in NR1 format
```

The :SBUS<n>:SPI:TRIGger:PATTern:MISO:WIDTh command sets the width of the SPI data pattern anywhere from 4 bits to 64 bits.

NOTE **The :SBUS<n>:SPI:TRIGger:PATTern:MISO:WIDTh should be set before :SBUS<n>:SPI:TRIGger:PATTern:MISO:DATA.**

Query Syntax

```
:SBUS<n>:SPI:TRIGger:PATTern:MISO:WIDTh?
```

The :SBUS<n>:SPI:TRIGger:PATTern:MISO:WIDTh? query returns the current SPI data pattern width setting.

Return Format

```
<width><NL>

<width> ::= integer from 4 to 64 in NR1 format
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":SBUS<n>:SPI:TRIGger:PATTern:MISO:DATA" on page 792
- ":SBUS<n>:SPI:SOURce:MISO" on page 790

:SBUS<n>:SPI:TRIGger:PATTern:MOSI:DATA

N (see page 1292)

Command Syntax

```
:SBUS<n>:SPI:TRIGger:PATTern:MOSI:DATA <string>

<string> ::= "nn...n" where n ::= {0 | 1 | X | $}

<string ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $}
```

The :SBUS<n>:SPI:TRIGger:PATTern:MOSI:DATA command defines the SPI data pattern resource according to the string parameter. This pattern, along with the data width, control the data pattern searched for in the data stream.

If the string parameter starts with "0x", it is a hexadecimal string made up of hexadecimal and X (don't care) characters; otherwise, it is a binary string made up of 0, 1, and X (don't care) characters.

NOTE **The :SBUS<n>:SPI:TRIGger:PATTern:MOSI:WIDTh should be set before :SBUS<n>:SPI:TRIGger:PATTern:MOSI:DATA.**

Query Syntax

```
:SBUS<n>:SPI:TRIGger:PATTern:MOSI:DATA?
```

The :SBUS<n>:SPI:TRIGger:PATTern:MOSI:DATA? query returns the current settings of the specified SPI data pattern resource in the binary string format.

Return Format

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

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":SBUS<n>:SPI:TRIGger:PATTern:MOSI:WIDTh" on page 795
- ":SBUS<n>:SPI:SOURce:MOSI" on page 791

:SBUS<n>:SPI:TRIGger:PATTern:MOSI:WIDTh

N (see page 1292)

Command Syntax

```
:SBUS<n>:SPI:TRIGger:PATTern:MOSI:WIDTh <width>

<width> ::= integer from 4 to 64 in NR1 format
```

The :SBUS<n>:SPI:TRIGger:PATTern:MOSI:WIDTh command sets the width of the SPI data pattern anywhere from 4 bits to 64 bits.

NOTE **The :SBUS<n>:SPI:TRIGger:PATTern:MOSI:WIDTh should be set before :SBUS<n>:SPI:TRIGger:PATTern:MOSI:DATA.**

Query Syntax

```
:SBUS<n>:SPI:TRIGger:PATTern:MOSI:WIDTh?
```

The :SBUS<n>:SPI:TRIGger:PATTern:MOSI:WIDTh? query returns the current SPI data pattern width setting.

Return Format

```
<width><NL>

<width> ::= integer from 4 to 64 in NR1 format
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":SBUS<n>:SPI:TRIGger:PATTern:MOSI:DATA" on page 794
- ":SBUS<n>:SPI:SOURce:MOSI" on page 791

:SBUS<n>:SPI:TRIGger:TYPE

N (see page 1292)

Command Syntax

```
:SBUS<n>:SPI:TRIGger:TYPE <value>

<value> ::= {MOSI | MISO}
```

The :SBUS<n>:SPI:TRIGger:TYPE command specifies whether the SPI trigger will be on the MOSI data or the MISO data.

When triggering on MOSI data, the data value is specified by the :SBUS<n>:SPI:TRIGger:PATTern:MOSI:DATA and :SBUS<n>:SPI:TRIGger:PATTern:MOSI:WIDTh commands.

When triggering on MISO data, the data value is specified by the :SBUS<n>:SPI:TRIGger:PATTern:MISO:DATA and :SBUS<n>:SPI:TRIGger:PATTern:MISO:WIDTh commands.

Query Syntax

```
:SBUS<n>:SPI:TRIGger:TYPE?
```

The :SBUS<n>:SPI:TRIGger:TYPE? query returns the current SPI trigger type setting.

Return Format

```
<value><NL>

<value> ::= {MOSI | MISO}
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":SBUS<n>:SPI:SOURce:MOSI" on page 791
- ":SBUS<n>:SPI:SOURce:MISO" on page 790
- ":SBUS<n>:SPI:TRIGger:PATTern:MISO:DATA" on page 792
- ":SBUS<n>:SPI:TRIGger:PATTern:MOSI:DATA" on page 794
- ":SBUS<n>:SPI:TRIGger:PATTern:MISO:WIDTh" on page 793
- ":SBUS<n>:SPI:TRIGger:PATTern:MOSI:WIDTh" on page 795
- ":TRIGger:MODE" on page 1091

:SBUS<n>:SPI:WIDTh

N (see page 1292)

Command Syntax

```
:SBUS<n>:SPI:WIDTh <word_width>

<word_width> ::= integer 4-16 in NR1 format
```

The :SBUS<n>:SPI:WIDTh command determines the number of bits in a word of data for SPI.

Query Syntax

```
:SBUS<n>:SPI:WIDTh?
```

The :SBUS<n>:SPI:WIDTh? query returns the current SPI decode word width.

Return Format

```
<word_width><NL>

<word_width> ::= integer 4-16 in NR1 format
```

Errors

- "-241, Hardware missing" on page 1247

See Also

- "Introduction to :SBUS<n> Commands" on page 715
- ":SBUS<n>:MODE" on page 719
- ":SBUS<n>:SPI Commands" on page 781

:SBUS<n>:UART Commands

NOTE These commands are only valid when the UART/RS-232 triggering and serial decode option has been licensed.

Table 100 :SBUS<n>:UART Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:UART:BASE <base> (see page 801)	:SBUS<n>:UART:BASE? (see page 801)	<base> ::= {ASCii | BINary | HEX}
:SBUS<n>:UART:BAUDrate <baudrate> (see page 802)	:SBUS<n>:UART:BAUDrate? (see page 802)	<baudrate> ::= integer from 100 to 8000000, 10000000, or 12000000
:SBUS<n>:UART:BITorder <bitorder> (see page 803)	:SBUS<n>:UART:BITorder? (see page 803)	<bitorder> ::= {LSBFirst | MSBFirst}
n/a	:SBUS<n>:UART:COUNt:ERRor? (see page 804)	<frame_count> ::= integer in NR1 format
:SBUS<n>:UART:COUNt:RESet (see page 805)	n/a	n/a
n/a	:SBUS<n>:UART:COUNt:RXFRames? (see page 806)	<frame_count> ::= integer in NR1 format
n/a	:SBUS<n>:UART:COUNt:TXFRames? (see page 807)	<frame_count> ::= integer in NR1 format
:SBUS<n>:UART:FRAMing <value> (see page 808)	:SBUS<n>:UART:FRAMing? (see page 808)	<value> ::= {OFF | <decimal> | <nondecimal>} <decimal> ::= 8-bit integer from 0-255 (0x00-0xff) <nondecimal> ::= #Hnn where n ::= {0,..,9 | A,..,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary
:SBUS<n>:UART:PARity <parity> (see page 809)	:SBUS<n>:UART:PARity? (see page 809)	<parity> ::= {EVEN | ODD | NONE}
:SBUS<n>:UART:POLarity <polarity> (see page 810)	:SBUS<n>:UART:POLarity? (see page 810)	<polarity> ::= {HIGH | LOW}

Table 100 :SBUS<n>:UART Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:UART:SOURce:RX <source> (see page 811)	:SBUS<n>:UART:SOURce:RX? (see page 811)	<source> ::= {CHANnel<n> | DIGital<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:SBUS<n>:UART:SOURce:TX <source> (see page 812)	:SBUS<n>:UART:SOURce:TX? (see page 812)	<source> ::= {CHANnel<n> | DIGital<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:SBUS<n>:UART:TRIGger:BASE <base> (see page 813)	:SBUS<n>:UART:TRIGger:BASE? (see page 813)	<base> ::= {ASCii | HEX}
:SBUS<n>:UART:TRIGger:BURSt <value> (see page 814)	:SBUS<n>:UART:TRIGger:BURSt? (see page 814)	<value> ::= {OFF | 1 to 4096 in NR1 format}
:SBUS<n>:UART:TRIGger:DATA <value> (see page 815)	:SBUS<n>:UART:TRIGger:DATA? (see page 815)	<value> ::= 8-bit integer from 0-255 (0x00-0xff) in decimal, <hexadecimal>, <binary>, or <quoted_string> format <hexadecimal> ::= #Hnn where n ::= {0,..,9 | A,..,F} for hexadecimal <binary> ::= #Bnn...n where n ::= {0 | 1} for binary <quoted_string> ::= any of the 128 valid 7-bit ASCII characters (or standard abbreviations)
:SBUS<n>:UART:TRIGger:IDLE <time_value> (see page 816)	:SBUS<n>:UART:TRIGger:IDLE? (see page 816)	<time_value> ::= time from 1 us to 10 s in NR3 format
:SBUS<n>:UART:TRIGger:QUALifier <value> (see page 817)	:SBUS<n>:UART:TRIGger:QUALifier? (see page 817)	<value> ::= {EQUal | NOTequal | GREaterthan | LESSthan}

Table 100 :SBUS<n>:UART Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:UART:TRIGger :TYPE <value> (see page 818)	:SBUS<n>:UART:TRIGger :TYPE? (see page 818)	<value> ::= {RSTArt | RSTOp | RDATa | RD1 | RD0 | RDX | RXPArity | TXPArity | TSTArt | TSTOp | TDATa | TD1 | TD0 | TDX}
:SBUS<n>:UART:WIDTh <width> (see page 819)	:SBUS<n>:UART:WIDTh? (see page 819)	<width> ::= {5 | 6 | 7 | 8 | 9}

:SBUS<n>:UART:BASE

(see page 1292)

Command Syntax

```
:SBUS<n>:UART:BASE <base>

<base> ::= {ASCii | BINary | HEX}
```

The :SBUS<n>:UART:BASE command determines the base to use for the UART decode and Lister display.

Query Syntax

```
:SBUS<n>:UART:BASE?
```

The :SBUS<n>:UART:BASE? query returns the current UART decode and Lister base setting.

Return Format

```
<base><NL>

<base> ::= {ASCii | BINary | HEX}
```

Errors

- "-241, Hardware missing" on page 1247

See Also

- "Introduction to :SBUS<n> Commands" on page 715
- ":SBUS<n>:UART Commands" on page 798

:SBUS<n>:UART:BAUDrate

N (see page 1292)

Command Syntax

```
:SBUS<n>:UART:BAUDrate <baudrate>

<baudrate> ::= integer from 100 to 8000000, 10000000, or 12000000
```

The :SBUS<n>:UART:BAUDrate command selects the bit rate (in bps) for the serial decoder and/or trigger when in UART mode. The baud rate can be set in the range from from 100 b/s to 8 Mb/s or to the specific values of 10 Mb/s or 12 Mb/s.

If the baud rate you select does not match the system baud rate, false triggers may occur.

Query Syntax

```
:SBUS<n>:UART:BAUDrate?
```

The :SBUS<n>:UART:BAUDrate? query returns the current UART baud rate setting.

Return Format

```
<baudrate><NL>

<baudrate> ::= integer from 100 to 8000000, 10000000, or 12000000
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:MODE" on page 1091
- ":SBUS<n>:UART:TRIGger:TYPE" on page 818

:SBUS<n>:UART:BITorder

(see page 1292)

Command Syntax

```
:SBUS<n>:UART:BITorder <bitorder>

<bitorder> ::= {LSBFirst | MSBFirst}
```

The :SBUS<n>:UART:BITorder command specifies the order of transmission used by the physical Tx and Rx input signals for the serial decoder and/or trigger when in UART mode. LSBFirst sets the least significant bit of each message "byte" as transmitted first. MSBFirst sets the most significant bit as transmitted first.

Query Syntax

```
:SBUS<n>:UART:BITorder?
```

The :SBUS<n>:UART:BITorder? query returns the current UART bit order setting.

Return Format

```
<bitorder><NL>

<bitorder> ::= {LSBF | MSBF}
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:MODE" on page 1091
- ":SBUS<n>:UART:TRIGger:TYPE" on page 818
- ":SBUS<n>:UART:SOURce:RX" on page 811
- ":SBUS<n>:UART:SOURce:TX" on page 812

:SBUS<n>:UART:COUNt:ERRor?

(see page 1292)

Query Syntax `:SBUS<n>:UART:COUNt:ERRor?`

Returns the UART error frame count.

Return Format `<frame_count><NL>`

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

Errors

- "-241, Hardware missing" on page 1247

See Also

- ":SBUS<n>:UART:COUNt:RESet" on page 805
- "Introduction to :SBUS<n> Commands" on page 715
- ":SBUS<n>:MODE" on page 719
- ":SBUS<n>:UART Commands" on page 798

:SBUS<n>:UART:COUNt:RESet

N (see page 1292)

Command Syntax `:SBUS<n>:UART:COUNt:RESet`

Resets the UART frame counters.

Errors

- "-241, Hardware missing" on page 1247

See Also

- ":SBUS<n>:UART:COUNt:ERRor?" on page 804
- ":SBUS<n>:UART:COUNt:RXFRames?" on page 806
- ":SBUS<n>:UART:COUNt:TXFRames?" on page 807
- "Introduction to :SBUS<n> Commands" on page 715
- ":SBUS<n>:MODE" on page 719
- ":SBUS<n>:UART Commands" on page 798

:SBUS<n>:UART:COUNt:RXFRames?

(see page 1292)

Query Syntax `:SBUS<n>:UART:COUNt:RXFRames?`

Returns the UART Rx frame count.

Return Format `<frame_count><NL>`

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

Errors

- "-241, Hardware missing" on page 1247

See Also

- ":SBUS<n>:UART:COUNt:RESet" on page 805
- "Introduction to :SBUS<n> Commands" on page 715
- ":SBUS<n>:MODE" on page 719
- ":SBUS<n>:UART Commands" on page 798

:SBUS<n>:UART:COUNt:TXFRames?

N (see page 1292)

Query Syntax `:SBUS<n>:UART:COUNt:TXFRames?`

Returns the UART Tx frame count.

Return Format `<frame_count><NL>`

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

Errors

- "-241, Hardware missing" on page 1247

See Also

- ":SBUS<n>:UART:COUNt:RESet" on page 805
- "Introduction to :SBUS<n> Commands" on page 715
- ":SBUS<n>:MODE" on page 719
- ":SBUS<n>:UART Commands" on page 798

:SBUS<n>:UART:FRAMing

N (see page 1292)

Command Syntax

```
:SBUS<n>:UART:FRAMing <value>

<value> ::= {OFF | <decimal> | <nondecimal>}

<decimal> ::= 8-bit integer in decimal from 0-255 (0x00-0xff)

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

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

The :SBUS<n>:UART:FRAMing command determines the byte value to use for framing (end of packet) or to turn off framing for UART decode.

Query Syntax

```
:SBUS<n>:UART:FRAMing?
```

The :SBUS<n>:UART:FRAMing? query returns the current UART decode base setting.

Return Format

```
<value><NL>

<value> ::= {OFF | <decimal>}

<decimal> ::= 8-bit integer in decimal from 0-255
```

Errors

- "-241, Hardware missing" on page 1247

See Also

- "Introduction to :SBUS<n> Commands" on page 715
- ":SBUS<n>:UART Commands" on page 798
- "Commands Not Supported in Multiple Program Message Units" on page 1297

:SBUS<n>:UART:PARity

(see page 1292)

Command Syntax

```
:SBUS<n>:UART:PARity <parity>

<parity> ::= {EVEN | ODD | NONE}
```

The :SBUS<n>:UART:PARity command selects the parity to be used with each message "byte" for the serial decoder and/or trigger when in UART mode.

Query Syntax

```
:SBUS<n>:UART:PARity?
```

The :SBUS<n>:UART:PARity? query returns the current UART parity setting.

Return Format

```
<parity><NL>

<parity> ::= {EVEN | ODD | NONE}
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:MODE" on page 1091
- ":SBUS<n>:UART:TRIGger:TYPE" on page 818

:SBUS<n>:UART:POLarity

N (see page 1292)

Command Syntax

```
:SBUS<n>:UART:POLarity <polarity>

<polarity> ::= {HIGH | LOW}
```

The :SBUS<n>:UART:POLarity command selects the polarity as idle low or idle high for the serial decoder and/or trigger when in UART mode.

Query Syntax

```
:SBUS<n>:UART:POLarity?
```

The :SBUS<n>:UART:POLarity? query returns the current UART polarity setting.

Return Format

```
<polarity><NL>

<polarity> ::= {HIGH | LOW}
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:MODE" on page 1091
- ":SBUS<n>:UART:TRIGger:TYPE" on page 818

:SBUS<n>:UART:SOURce:RX

(see page 1292)

Command Syntax

```
:SBUS<n>:UART:SOURce:RX <source>

<source> ::= {CHANnel<n> | DIGital<d>}

<n> ::= 1 to (# analog channels) in NR1 format

<d> ::= 0 to (# digital channels - 1) in NR1 format
```

The :SBUS<n>:UART:SOURce:RX command controls which signal is used as the Rx source by the serial decoder and/or trigger when in UART mode.

Query Syntax

```
:SBUS<n>:UART:SOURce:RX?
```

The :SBUS<n>:UART:SOURce:RX? query returns the current source for the UART Rx signal.

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:MODE" on page 1091
- ":SBUS<n>:UART:TRIGger:TYPE" on page 818
- ":SBUS<n>:UART:BITorder" on page 803

:SBUS<n>:UART:SOURce:TX

 (see page 1292)

Command Syntax

```
:SBUS<n>:UART:SOURce:TX <source>

<source> ::= {CHANnel<n> | DIGital<d>}

<n> ::= 1 to (# analog channels) in NR1 format

<d> ::= 0 to (# digital channels - 1) in NR1 format
```

The :SBUS<n>:UART:SOURce:TX command controls which signal is used as the Tx source by the serial decoder and/or trigger when in UART mode.

Query Syntax

```
:SBUS<n>:UART:SOURce:TX?
```

The :SBUS<n>:UART:SOURce:TX? query returns the current source for the UART Tx signal.

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:MODE" on page 1091
- ":SBUS<n>:UART:TRIGger:TYPE" on page 818
- ":SBUS<n>:UART:BITorder" on page 803

:SBUS<n>:UART:TRIGger:BASE

(see page 1292)

Command Syntax

```
:SBUS<n>:UART:TRIGger:BASE <base>

<base> ::= {ASCii | HEX}
```

The :SBUS<n>:UART:TRIGger:BASE command sets the front panel UART/RS232 trigger setup data selection option:

- ASCii – front panel data selection is from ASCII values.
- HEX – front panel data selection is from hexadecimal values.

The :SBUS<n>:UART:TRIGger:BASE setting does not affect the :SBUS<n>:UART:TRIGger:DATA command which can always set data values using ASCII or hexadecimal values.

NOTE **The :SBUS<n>:UART:TRIGger:BASE command is independent of the :SBUS<n>:UART:BASE command which affects decode and Lister only.**

Query Syntax

```
:SBUS<n>:UART:TRIGger:BASE?
```

The :SBUS<n>:UART:TRIGger:BASE? query returns the current UART base setting.

Return Format

```
<base><NL>

<base> ::= {ASC | HEX}
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:MODE" on page 1091
- ":SBUS<n>:UART:TRIGger:DATA" on page 815

:SBUS<n>:UART:TRIGger:BURSt

 (see page 1292)

Command Syntax

```
:SBUS<n>:UART:TRIGger:BURSt <value>

<value> ::= {OFF | 1 to 4096 in NR1 format}
```

The :SBUS<n>:UART:TRIGger:BURSt command selects the burst value (Nth frame after idle period) in the range 1 to 4096 or OFF, for the trigger when in UART mode.

Query Syntax

```
:SBUS<n>:UART:TRIGger:BURSt?
```

The :SBUS<n>:UART:TRIGger:BURSt? query returns the current UART trigger burst value.

Return Format

```
<value><NL>

<value> ::= {OFF | 1 to 4096 in NR1 format}
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:MODE" on page 1091
- ":SBUS<n>:UART:TRIGger:IDLE" on page 816
- ":SBUS<n>:UART:TRIGger:TYPE" on page 818

:SBUS<n>:UART:TRIGger:DATA

N (see page 1292)

Command Syntax

```
:SBUS<n>:UART:TRIGger:DATA <value>

<value> ::= 8-bit integer from 0-255 (0x00-0xff) in decimal,
            <hexadecimal>, <binary>, or <quoted_string> format

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

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

<quoted_string> ::= any of the 128 valid 7-bit ASCII characters
                    (or standard abbreviations)
```

The :SBUS<n>:UART:TRIGger:DATA command selects the data byte value (0x00 to 0xFF) for the trigger QUALifier when in UART mode. The data value is used when one of the RD or TD trigger types is selected.

When entering an ASCII character via the quoted string, it must be one of the 128 valid characters (case-sensitive): "NUL", "SOH", "STX", "ETX", "EOT", "ENQ", "ACK", "BEL", "BS", "HT", "LF", "VT", "FF", "CR", "SO","SI", "DLE", "DC1", "DC2", "DC3", "DC4", "NAK", "SYN", "ETB", "CAN", "EM", "SUB", "ESC", "FS","GS", "RS", "US", "SP", "!", "\"", "#", "$", "%","&", "\'", "(", ")", "*", "+", ",", "-", ".", "/","0", "1", "2", "3", "4", "5", "6", "7", "8", "9",":", ";", "<", "=", ">", "?", "@", "A", "B", "C","D", "E", "F", "G", "H", "I", "J", "K", "L", "M","N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z","[", "\\", "]", "^", "_", "`", "a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m", "n", "o", "p", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z", "{", "|", "}", "~", or "DEL".

Query Syntax

```
:SBUS<n>:UART:TRIGger:DATA?
```

The :SBUS<n>:UART:TRIGger:DATA? query returns the current UART trigger data value.

Return Format

```
<value><NL>

<value> ::= 8-bit integer in decimal from 0-255
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:MODE" on page 1091
- ":SBUS<n>:UART:TRIGger:BASE" on page 813
- ":SBUS<n>:UART:TRIGger:TYPE" on page 818

:SBUS<n>:UART:TRIGger:IDLE

 (see page 1292)

Command Syntax

```
:SBUS<n>:UART:TRIGger:IDLE <time_value>

<time_value> ::= time from 1 us to 10 s in NR3 format
```

The :SBUS<n>:UART:TRIGger:IDLE command selects the value of the idle period for burst trigger in the range from 1 us to 10 s when in UART mode.

Query Syntax

```
:SBUS<n>:UART:TRIGger:IDLE?
```

The :SBUS<n>:UART:TRIGger:IDLE? query returns the current UART trigger idle period time.

Return Format

```
<time_value><NL>

<time_value> ::= time from 1 us to 10 s in NR3 format
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:MODE" on page 1091
- ":SBUS<n>:UART:TRIGger:BURSt" on page 814
- ":SBUS<n>:UART:TRIGger:TYPE" on page 818

:SBUS<n>:UART:TRIGger:QUALifier

N (see page 1292)

Command Syntax

```
:SBUS<n>:UART:TRIGger:QUALifier <value>

<value> ::= {EQUal | NOTequal | GREaterthan | LESSthan}
```

The :SBUS<n>:UART:TRIGger:QUALifier command selects the data qualifier when :TYPE is set to RDATa, RD1, RD0, RDX, TDATa, TD1, TD0, or TDX for the trigger when in UART mode.

Query Syntax

```
:SBUS<n>:UART:TRIGger:QUALifier?
```

The :SBUS<n>:UART:TRIGger:QUALifier? query returns the current UART trigger qualifier.

Return Format

```
<value><NL>

<value> ::= {EQU | NOT | GRE | LESS}
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:MODE" on page 1091
- ":SBUS<n>:UART:TRIGger:TYPE" on page 818

:SBUS<n>:UART:TRIGger:TYPE

N (see page 1292)

Command Syntax

```
:SBUS<n>:UART:TRIGger:TYPE <value>

<value> ::= {RSTArt | RSTOp | RDATa | RD1 | RD0 | RDX | RXPArity
   | TXPArity | TSTArt | TSTOp | TDATa | TD1 | TD0 | TDX}
```

The :SBUS<n>:UART:TRIGger:TYPE command selects the UART trigger type.

When one of the RD or TD types is selected, the :SBUS<n>:UART:TRIGger:DATA and :SBUS<n>:UART:TRIGger:QUALifier commands are used to specify the data value and comparison operator.

The RD1, RD0, RDX, TD1, TD0, and TDX types (for triggering on data and alert bit values) are only valid when a 9-bit width has been selected.

Query Syntax

```
:SBUS<n>:UART:TRIGger:TYPE?
```

The :SBUS<n>:UART:TRIGger:TYPE? query returns the current UART trigger data value.

Return Format

```
<value><NL>

<value> ::= {RSTA | RSTO | RDAT | RD1 | RD0 | RDX | RXPA | TXPA | TSTA
   | TSTO | TDAT | TD1 | TD0 | TDX}
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:MODE" on page 1091
- ":SBUS<n>:UART:TRIGger:DATA" on page 815
- ":SBUS<n>:UART:TRIGger:QUALifier" on page 817
- ":SBUS<n>:UART:WIDTh" on page 819

:SBUS<n>:UART:WIDTh

N (see page 1292)

Command Syntax

```
:SBUS<n>:UART:WIDTh <width>

<width> ::= {5 | 6 | 7 | 8 | 9}
```

The :SBUS<n>:UART:WIDTh command determines the number of bits (5-9) for each message "byte" for the serial decoder and/or trigger when in UART mode.

Query Syntax

```
:SBUS<n>:UART:WIDTh?
```

The :SBUS<n>:UART:WIDTh? query returns the current UART width setting.

Return Format

```
<width><NL>

<width> ::= {5 | 6 | 7 | 8 | 9}
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:MODE" on page 1091
- ":SBUS<n>:UART:TRIGger:TYPE" on page 818

31 :SEARch Commands

Control the event search modes and parameters for each search type. See:

General :SEARch Commands

Table 101 General :SEARch Commands Summary

Command	Query	Options and Query Returns
n/a	:SEARch:COUNt? (see page 823)	<count> ::= an integer count value
:SEARch:EVENt <event_number> (see page 824)	:SEARch:EVENt? (see page 824)	<event_number> ::= the integer number of a found search event
:SEARch:MODE <value> (see page 825)	:SEARch:MODE? (see page 825)	<value> ::= {EDGE | GLITch | TRANsition | SERial{1 | 2} | PEAK}
:SEARch:STATe <value> (see page 826)	:SEARch:STATe? (see page 826)	<value> ::= {{0 | OFF} | {1 | ON}}

:SEARch:COUNt?

(see page 1292)

Query Syntax `:SEARch:COUNt?`

The :SEARch:COUNt? query returns the number of search events found.

Return Format `<count><NL>`

`<count> ::= an integer count value`

See Also

- Chapter 31, ":SEARch Commands," starting on page 821
- ":SEARch:EVENt" on page 824
- ":SEARch:STATe" on page 826
- ":SEARch:MODE" on page 825

:SEARch:EVENt

N (see page 1292)

Command Syntax

```
:SEARch:EVENt <event_number>

<event_number> ::= the integer number of a found search event
```

The :SEARch:EVENt command navigates to a found search event.

If the :SEARch:STATe is ON, the horizontal position is changed so that the specified event is located at the time reference.

Query Syntax

```
:SEARch:EVENt?
```

The :SEARch:EVENt? query returns the currently selected event number.

Return Format

```
<event_number><NL>

<event_number> ::= the integer number of a found search event
```

See Also

- Chapter 31, ":SEARch Commands," starting on page 821
- ":SEARch:COUNt?" on page 823
- ":SEARch:STATe" on page 826
- ":SEARch:MODE" on page 825

:SEARch:MODE

(see page 1292)

Command Syntax

```
:SEARch:MODE <value>

<value> ::= {EDGE | GLITch | TRANsition | SERial{1 | 2} | PEAK}
```

The :SEARch:MODE command selects the search mode.

The command is only valid when the :SEARch:STATe is ON.

Query Syntax

```
:SEARch:MODE?
```

The :SEARch:MODE? query returns the currently selected mode or OFF if the :SEARch:STATe is OFF.

Return Format

```
<value><NL>

<value> ::= {EDGE | GLIT | TRAN | SER{1 | 2} | PEAK | OFF}
```

See Also

- Chapter 31, ":SEARch Commands," starting on page 821
- ":SEARch:STATe" on page 826
- ":SEARch:COUNt?" on page 823
- ":SEARch:EVENt" on page 824

:SEARch:STATe

 (see page 1292)

Command Syntax

```
:SEARch:STATe <value>

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

The :SEARch:STATe command enables or disables the search feature.

Query Syntax

```
:SEARch:STATe?
```

The :SEARch:STATe? query returns returns the current setting.

Return Format

```
<value><NL>

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

See Also

- Chapter 31, ":SEARch Commands," starting on page 821
- ":SEARch:MODE" on page 825
- ":SEARch:COUNt?" on page 823
- ":SEARch:EVENt" on page 824

:SEARch:EDGE Commands

Table 102 :SEARch:EDGE Commands Summary

Command	Query	Options and Query Returns
:SEARch:EDGE:SLOPe <slope> (see page 828)	:SEARch:EDGE:SLOPe? (see page 828)	<slope> ::= {POSitive | NEGative | EITHer}
:SEARch:EDGE:SOURce <source> (see page 829)	:SEARch:EDGE:SOURce? (see page 829)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format

:SEARch:EDGE:SLOPe

N (see page 1292)

Command Syntax

```
:SEARch:EDGE:SLOPe <slope>

<slope> ::= {NEGative | POSitive | EITHer}
```

The :SEARch:EDGE:SLOPe command specifies the slope of the edge for the search.

Query Syntax

```
:SEARch:EDGE:SLOPe?
```

The :SEARch:EDGE:SLOPe? query returns the current slope setting.

Return Format

```
<slope><NL>

<slope> ::= {NEG | POS | EITH}
```

See Also

- Chapter 31, ":SEARch Commands," starting on page 821

:SEARch:EDGE:SOURce

 (see page 1292)

Command Syntax

```
:SEARch:EDGE:SOURce <source>

<source> ::= CHANnel<n>

<n> ::= 1 to (# analog channels) in NR1 format
```

The :SEARch:EDGE:SOURce command selects the channel on which to search for edges.

Query Syntax

```
:SEARch:EDGE:SOURce?
```

The :SEARch:EDGE:SOURce? query returns the current source.

Return Format

```
<source><NL>

<source> ::= CHAN<n>
```

See Also

- Chapter 31, ":SEARch Commands," starting on page 821

:SEARch:GLITch Commands

Table 103 :SEARch:GLITch Commands Summary

Command	Query	Options and Query Returns
:SEARch:GLITch:GREaterthan <greater_than_time>[suffix] (see page 831)	:SEARch:GLITch:GREaterthan? (see page 831)	<greater_than_time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:SEARch:GLITch:LESSthan <less_than_time>[suffix] (see page 832)	:SEARch:GLITch:LESSthan? (see page 832)	<less_than_time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:SEARch:GLITch:POLarity <polarity> (see page 833)	:SEARch:GLITch:POLarity? (see page 833)	<polarity> ::= {POSitive | NEGative}
:SEARch:GLITch:QUALifier <qualifier> (see page 834)	:SEARch:GLITch:QUALifier? (see page 834)	<qualifier> ::= {GREaterthan | LESSthan | RANGe}
:SEARch:GLITch:RANGe <less_than_time>[suffix], <greater_than_time>[suffix] (see page 835)	:SEARch:GLITch:RANGe? (see page 835)	<less_than_time> ::= 15 ns to 10 seconds in NR3 format <greater_than_time> ::= 10 ns to 9.99 seconds in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:SEARch:GLITch:SOURce <source> (see page 836)	:SEARch:GLITch:SOURce? (see page 836)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format

:SEARch:GLITch:GREaterthan

(see page 1292)

Command Syntax

```
:SEARch:GLITch:GREaterthan <greater_than_time>[<suffix>]

<greater_than_time> ::= floating-point number in NR3 format

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

The :SEARch:GLITch:GREaterthan command sets the minimum pulse width duration for the selected :SEARch:GLITch:SOURce.

Query Syntax

```
:SEARch:GLITch:GREaterthan?
```

The :SEARch:GLITch:GREaterthan? query returns the minimum pulse width duration time for :SEARch:GLITch:SOURce.

Return Format

```
<greater_than_time><NL>

<greater_than_time> ::= floating-point number in NR3 format.
```

See Also

- Chapter 31, ":SEARch Commands," starting on page 821
- ":SEARch:GLITch:SOURce" on page 836
- ":SEARch:GLITch:QUALifier" on page 834
- ":SEARch:MODE" on page 825

:SEARch:GLITch:LESSthan

N (see page 1292)

Command Syntax

```
:SEARch:GLITch:LESSthan <less_than_time>[<suffix>]

<less_than_time> ::= floating-point number in NR3 format

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

The :SEARch:GLITch:LESSthan command sets the maximum pulse width duration for the selected :SEARch:GLITch:SOURce.

Query Syntax

```
:SEARch:GLITch:LESSthan?
```

The :SEARch:GLITch:LESSthan? query returns the pulse width duration time for :SEARch:GLITch:SOURce.

Return Format

```
<less_than_time><NL>

<less_than_time> ::= floating-point number in NR3 format.
```

See Also

- Chapter 31, ":SEARch Commands," starting on page 821
- ":SEARch:GLITch:SOURce" on page 836
- ":SEARch:GLITch:QUALifier" on page 834
- ":SEARch:MODE" on page 825

:SEARch:GLITch:POLarity

(see page 1292)

Command Syntax

```
:SEARch:GLITch:POLarity <polarity>

<polarity> ::= {POSitive | NEGative}
```

The :SEARch:GLITch:POLarity command sets the polarity for the glitch (pulse width) search.

Query Syntax

```
:SEARch:GLITch:POLarity?
```

The :SEARch:GLITch:POLarity? query returns the current polarity setting for the glitch (pulse width) search.

Return Format

```
<polarity><NL>

<polarity> ::= {POS | NEG}
```

See Also

- Chapter 31, ":SEARch Commands," starting on page 821
- ":SEARch:MODE" on page 825
- ":SEARch:GLITch:SOURce" on page 836

:SEARch:GLITch:QUALifier

(see page 1292)

Command Syntax

```
:SEARch:GLITch:QUALifier <operator>

<operator> ::= {GREaterthan | LESSthan | RANGe}
```

This command sets the mode of operation of the glitch (pulse width) search. The oscilloscope can search for a pulse width that is greater than a time value, less than a time value, or within a range of time values.

Query Syntax

```
:SEARch:GLITch:QUALifier?
```

The :SEARch:GLITch:QUALifier? query returns the glitch (pulse width) qualifier.

Return Format

```
<operator><NL>

<operator> ::= {GRE | LESS | RANG}
```

See Also

- Chapter 31, ":SEARch Commands," starting on page 821
- ":SEARch:GLITch:SOURce" on page 836
- ":SEARch:MODE" on page 825

:SEARch:GLITch:RANGe

N (see page 1292)

Command Syntax

```
:SEARch:GLITch:RANGe <less_than_time>[suffix],
                     <greater_than_time>[suffix]

<less_than_time> ::= (15 ns - 10 seconds) in NR3 format

<greater_than_time> ::= (10 ns - 9.99 seconds) in NR3 format

[suffix] ::= {s | ms | us | ns | ps}
```

The :SEARch:GLITch:RANGe command sets the pulse width duration for the selected :SEARch:GLITch:SOURce. You can enter the parameters in any order — the smaller value becomes the <greater_than_time> and the larger value becomes the <less_than_time>.

Query Syntax

```
:SEARch:GLITch:RANGe?
```

The :SEARch:GLITch:RANGe? query returns the pulse width duration time for :SEARch:GLITch:SOURce.

Return Format

```
<less_than_time>,<greater_than_time><NL>
```

See Also

- Chapter 31, ":SEARch Commands," starting on page 821
- ":SEARch:GLITch:SOURce" on page 836
- ":SEARch:GLITch:QUALifier" on page 834
- ":SEARch:MODE" on page 825

:SEARch:GLITch:SOURce

(see page 1292)

Command Syntax

```
:SEARch:GLITch:SOURce <source>

<source> ::= CHANnel<n>

<n> ::= 1 to (# analog channels) in NR1 format
```

The :SEARch:GLITch:SOURce command selects the channel on which to search for glitches (pulse widths).

Query Syntax

```
:SEARch:GLITch:SOURce?
```

The :SEARch:GLITch:SOURce? query returns the current pulse width source.

If all channels are off, the query returns "NONE."

Return Format

```
<source><NL>
```

See Also

- Chapter 31, ":SEARch Commands," starting on page 821
- Chapter 31, ":SEARch Commands," starting on page 821
- ":SEARch:MODE" on page 825
- ":SEARch:GLITch:POLarity" on page 833
- ":SEARch:GLITch:QUALifier" on page 834
- ":SEARch:GLITch:RANGe" on page 835

:SEARch:PEAK Commands

Table 104 :SEARch:PEAK Commands Summary

Command	Query	Options and Query Returns
:SEARch:PEAK:EXCursion <delta_level> (see page 838)	:SEARch:PEAK:EXCursion? (see page 838)	<delta_level> ::= required change in level to be recognized as a peak, in NR3 format.
:SEARch:PEAK:NPEaks <number> (see page 839)	:SEARch:PEAK:NPEaks? (see page 839)	<number> ::= max number of peaks to find, 1-11 in NR1 format.
:SEARch:PEAK:SOURce <source> (see page 840)	:SEARch:PEAK:SOURce? (see page 840)	<source> ::= {FUNCtion<m> | MATH<m> | FFT} (must be an FFT waveform) <m> ::= 1 to (# math functions) in NR1 format
:SEARch:PEAK:THReshold <level> (see page 841)	:SEARch:PEAK:THReshold? (see page 841)	<level> ::= necessary level to be considered a peak, in NR3 format.

:SEARch:PEAK:EXCursion

(see page 1292)

Command Syntax

```
:SEARch:PEAK:EXCursion <delta_level>

<delta_level> ::= required change in level to be recognized as a peak,
   in NR3 format.
```

The :SEARch:PEAK:EXCursion command specifies the change in level that must occur (in other words, hysteresis) to be recognized as a peak.

The threshold level units are specified by the :FFT:VTYPe or :FUNCtion<m>[:FFT]:VTYPe command.

Query Syntax

```
:SEARch:PEAK:EXCursion?
```

The :SEARch:PEAK:EXCursion? query returns the specified excursion delta level value.

Return Format

```
<delta_level><NL>

<delta_level> ::= in NR3 format.
```

See Also

- ":FFT:VTYPe" on page 398
- ":FUNCtion<m>[:FFT]:VTYPe" on page 444
- ":SEARch:PEAK:NPEaks" on page 839
- ":SEARch:PEAK:SOURce" on page 840
- ":SEARch:PEAK:THReshold" on page 841

:SEARch:PEAK:NPEaks

(see page 1292)

Command Syntax

```
:SEARch:PEAK:NPEaks <number>

<number> ::= max number of peaks to find, 1-11 in NR1 format.
```

The :SEARch:PEAK:NPEaks command specifies the maximum number of FFT peaks to find. This number can be from 1 to 11.

Query Syntax

```
:SEARch:PEAK:NPEaks?
```

The :SEARch:PEAK:NPEaks? query returns the specified maximum number of FFT peaks to find.

Return Format

```
<number><NL>

<number> ::= in NR1 format.
```

See Also

- ":SEARch:PEAK:EXCursion" on page 838
- ":SEARch:PEAK:SOURce" on page 840
- ":SEARch:PEAK:THReshold" on page 841

:SEARch:PEAK:SOURce

(see page 1292)

Command Syntax

```
:SEARch:PEAK:SOURce <source>

<source> ::= {FUNCtion<m> | MATH<m> | FFT} (source must be
   an FFT waveform)

<m> ::= 1 to (# math functions) in NR1 format
```

The :SEARch:PEAK:SOURce command selects the FFT math function waveform to search.

Query Syntax

```
:SEARch:PEAK:SOURce?
```

The :SEARch:PEAK:SOURce? query returns the FFT math function waveform that is being searched.

Return Format

```
<source><NL>

<source> ::= {FUNC<m> | FFT} (must be FFT)

<m> ::= 1 to (# math functions) in NR1 format
```

See Also

- ":SEARch:PEAK:EXCursion" on page 838
- ":SEARch:PEAK:NPEaks" on page 839
- ":SEARch:PEAK:THReshold" on page 841

:SEARch:PEAK:THReshold

(see page 1292)

Command Syntax

```
:SEARch:PEAK:THReshold <level>

<level> ::= necessary level to be considered a peak, in NR3 format.
```

The :SEARch:PEAK:THReshold command specifies the threshold level necessary to be considered a peak.

The threshold level units are specified by the :FFT:VTYPe or :FUNCtion<m>[:FFT]:VTYPe command.

Query Syntax

```
:SEARch:PEAK:THReshold?
```

The :SEARch:PEAK:THReshold? query returns the specified threshold level for FFT peak search.

Return Format

```
<level><NL>

<level> ::= in NR3 format.
```

See Also

- ":FFT:VTYPe" on page 398
- ":FUNCtion<m>[:FFT]:VTYPe" on page 444
- ":SEARch:PEAK:EXCursion" on page 838
- ":SEARch:PEAK:NPEaks" on page 839
- ":SEARch:PEAK:SOURce" on page 840

:SEARch:TRANsition Commands

Table 105 :SEARch:TRANsition Commands Summary

Command	Query	Options and Query Returns
:SEARch:TRANsition:QUALifier <qualifier> (see page 843)	:SEARch:TRANsition:QUALifier? (see page 843)	<qualifier> ::= {GREaterthan | LESSthan}
:SEARch:TRANsition:SLOPe <slope> (see page 844)	:SEARch:TRANsition:SLOPe? (see page 844)	<slope> ::= {NEGative | POSitive}
:SEARch:TRANsition:SOURce <source> (see page 845)	:SEARch:TRANsition:SOURce? (see page 845)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:SEARch:TRANsition:TIME <time>[suffix] (see page 846)	:SEARch:TRANsition:TIME? (see page 846)	<time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}

:SEARch:TRANsition:QUALifier

 (see page 1292)

Command Syntax

```
:SEARch:TRANsition:QUALifier <qualifier>

<qualifier> ::= {GREaterthan | LESSthan}
```

The :SEARch:TRANsition:QUALifier command specifies whether to search for edge transitions greater than or less than a time.

Query Syntax

```
:SEARch:TRANsition:QUALifier?
```

The :SEARch:TRANsition:QUALifier? query returns the current transition search qualifier.

Return Format

```
<qualifier><NL>

<qualifier> ::= {GRE | LESS}
```

See Also

- Chapter 31, ":SEARch Commands," starting on page 821
- ":SEARch:MODE" on page 825
- ":SEARch:TRANsition:TIME" on page 846

:SEARch:TRANsition:SLOPe

N (see page 1292)

Command Syntax

```
:SEARch:TRANsition:SLOPe <slope>

<slope> ::= {NEGative | POSitive}
```

The :SEARch:TRANsition:SLOPe command selects whether to search for rising edge (POSitive slope) transitions or falling edge (NEGative slope) transitions.

Query Syntax

```
:SEARch:TRANsition:SLOPe?
```

The :SEARch:TRANsition:SLOPe? query returns the current transition search slope setting.

Return Format

```
<slope><NL>

<slope> ::= {NEG | POS}
```

See Also

- Chapter 31, ":SEARch Commands," starting on page 821
- ":SEARch:MODE" on page 825
- ":SEARch:TRANsition:SOURce" on page 845
- ":SEARch:TRANsition:TIME" on page 846

:SEARch:TRANsition:SOURce

 (see page 1292)

Command Syntax

```
:SEARch:TRANsition:SOURce <source>

<source> ::= CHANnel<n>

<n> ::= 1 to (# analog channels) in NR1 format
```

The :SEARch:TRANsition:SOURce command selects the channel on which to search for edge transitions.

Query Syntax

```
:SEARch:TRANsition:SOURce?
```

The :SEARch:TRANsition:SOURce? query returns the current transition search source.

Return Format

```
<source><NL>

<source> ::= CHAN<n>
```

See Also

- Chapter 31, ":SEARch Commands," starting on page 821
- ":SEARch:MODE" on page 825
- ":SEARch:TRANsition:SLOPe" on page 844

:SEARch:TRANsition:TIME

 (see page 1292)

Command Syntax

```
:SEARch:TRANsition:TIME <time>[suffix]

<time> ::= floating-point number in NR3 format

[suffix] ::= {s | ms | us | ns | ps}
```

The :SEARch:TRANsition:TIME command sets the time of the transition to search for. You can search for transitions greater than or less than this time.

Query Syntax

```
:SEARch:TRANsition:TIME?
```

The :SEARch:TRANsition:TIME? query returns the current transition time value.

Return Format

```
<time><NL>

<time> ::= floating-point number in NR3 format
```

See Also

- Chapter 31, ":SEARch Commands," starting on page 821
- ":SEARch:TRANsition:QUALifier" on page 843

:SEARch:SERial:CAN Commands

Table 106 :SEARch:SERial:CAN Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:CAN:MODE <value> (see page 848)	:SEARch:SERial:CAN:MODE? (see page 848)	<value> ::= {IDEither | IDData | DATA | IDRemote | ERRor | ACKerror | FORMerror | STUFferror | CRCerror | ALLerrors | OVERload | MESSage | MSIGnal}
:SEARch:SERial:CAN:PATTern:DATA <string> (see page 850)	:SEARch:SERial:CAN:PATTern:DATA? (see page 850)	<string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X} for hexadecimal
:SEARch:SERial:CAN:PATTern:DATA:LENGth <length> (see page 851)	:SEARch:SERial:CAN:PATTern:DATA:LENGth? (see page 851)	<length> ::= integer from 1 to 8 in NR1 format
:SEARch:SERial:CAN:PATTern:ID <string> (see page 852)	:SEARch:SERial:CAN:PATTern:ID? (see page 852)	<string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X} for hexadecimal
:SEARch:SERial:CAN:PATTern:ID:MODE <value> (see page 853)	:SEARch:SERial:CAN:PATTern:ID:MODE? (see page 853)	<value> ::= {STANdard | EXTended}
:SEARch:SERial:CAN:SYMBolic:MESSage <name> (see page 854)	:SEARch:SERial:CAN:SYMBolic:MESSage? (see page 854)	<name> ::= quoted ASCII string
:SEARch:SERial:CAN:SYMBolic:SIGNal <name> (see page 855)	:SEARch:SERial:CAN:SYMBolic:SIGNal? (see page 855)	<name> ::= quoted ASCII string
:SEARch:SERial:CAN:SYMBolic:VALue <data> (see page 856)	:SEARch:SERial:CAN:SYMBolic:VALue? (see page 856)	<data> ::= value in NR3 format

:SEARch:SERial:CAN:MODE

N (see page 1292)

Command Syntax

```
:SEARch:SERial:CAN:MODE <value>

<value> ::= {IDEither | IDData | DATA | IDRemote | ERRor | ACKerror
   | FORMerror | STUFferror | CRCerror | ALLerrors | OVERload
   | MESSage | MSIGnal}
```

The :SEARch:SERial:CAN:MODE command selects the type of CAN information to find in the Lister display:

Condition	Front-panel name	Description
IDEither	Frame ID	Finds remote or data frames matching the specified ID.
IDData	Data Frame ID	Finds data frames matching the specified ID.
DATA	Data Frame ID and Data	Finds data frames matching the specified ID and data.
IDRemote	Remote Frame ID	Finds remote frames with the specified ID.
ERRor	Error Frame	Finds CAN active error frames.
ACKerror	Acknowledge Error	Finds the acknowledge bit if the polarity is incorrect.
FORMerror	Form Error	Finds reserved bit errors.
STUFferror	Stuff Error	Finds 6 consecutive 1s or 6 consecutive 0s, while in a non-error or non overload frame.
CRCerror	CRC Field Error	Finds when the calculated CRC does not match the transmitted CRC.
ALLerrors	All Errors	Finds any form error or active error.
OVERload	Overload Frame	Finds CAN overload frames.
MESSage	Message	Finds a symbolic message.
MSIGnal	Message and Signal (non-FD)	Finds a symbolic message and a signal value.

Query Syntax

```
:SEARch:SERial:CAN:MODE?
```

The :SEARch:SERial:CAN:MODE? query returns the currently selected mode.

Return Format

```
<value><NL>

<value> ::= {IDE | IDD | DATA | IDR | ERR | ACK | FORM | STUF | CRC
   | ALL | OVER | MESS | MSIG}
```

See Also

- Chapter 31, ":SEARch Commands," starting on page 821
- ":SEARch:SERial:CAN:PATTern:DATA" on page 850
- ":SEARch:SERial:CAN:PATTern:ID" on page 852

- ":RECall:DBC[:STARt]" on page 679
- ":SEARch:SERial:CAN:SYMBolic:MESSage" on page 854
- ":SEARch:SERial:CAN:SYMBolic:SIGNal" on page 855
- ":SEARch:SERial:CAN:SYMBolic:VALue" on page 856

:SEARch:SERial:CAN:PATTern:DATA

 (see page 1292)

Command Syntax

```
:SEARch:SERial:CAN:PATTern:DATA <string>

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

The :SEARch:SERial:CAN:PATTern:DATA command specifies the data value when searching for Data Frame ID and Data.

The length of the data value is specified using the :SEARch:SERial:CAN:PATTern:DATA:LENGth command.

Query Syntax

```
:SEARch:SERial:CAN:PATTern:DATA?
```

The :SEARch:SERial:CAN:PATTern:DATA? query returns the current data value setting.

Return Format

```
<string><NL>

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

See Also

- Chapter 31, ":SEARch Commands," starting on page 821
- ":SEARch:SERial:CAN:MODE" on page 848
- ":SEARch:SERial:CAN:PATTern:DATA:LENGth" on page 851

:SEARch:SERial:CAN:PATTern:DATA:LENGth

 (see page 1292)

Command Syntax

```
:SEARch:SERial:CAN:PATTern:DATA:LENGth <length>

<length> ::= integer from 1 to 8 in NR1 format
```

The :SEARch:SERial:CAN:PATTern:DATA:LENGth command specifies the length of the data value when searching for Data Frame ID and Data.

The data value is specified using the :SEARch:SERial:CAN:PATTern:DATA command.

Query Syntax

```
:SEARch:SERial:CAN:PATTern:DATA:LENGth?
```

The :SEARch:SERial:CAN:PATTern:DATA:LENGth? query returns the current data length setting.

Return Format

```
<length><NL>

<length> ::= integer from 1 to 8 in NR1 format
```

See Also

- Chapter 31, ":SEARch Commands," starting on page 821
- ":SEARch:SERial:CAN:MODE" on page 848
- ":SEARch:SERial:CAN:PATTern:DATA" on page 850

:SEARch:SERial:CAN:PATTern:ID

N (see page 1292)

Command Syntax

```
:SEARch:SERial:CAN:PATTern:ID <string>

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

The :SEARch:SERial:CAN:PATTern:ID command specifies the ID value when searching for a CAN event.

The value can be a standard ID or an extended ID, depending on the :SEARch:SERial:CAN:PATTern:ID:MODE command's setting.

Query Syntax

```
:SEARch:SERial:CAN:PATTern:ID?
```

The :SEARch:SERial:CAN:PATTern:ID? query returns the current ID value setting.

Return Format

```
<string><NL>

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

See Also

- Chapter 31, ":SEARch Commands," starting on page 821
- ":SEARch:SERial:CAN:MODE" on page 848
- ":SEARch:SERial:CAN:PATTern:ID:MODE" on page 853

:SEARch:SERial:CAN:PATTern:ID:MODE

 (see page 1292)

Command Syntax

```
:SEARch:SERial:CAN:PATTern:ID:MODE <value>

<value> ::= {STANdard | EXTended}
```

The :SEARch:SERial:CAN:PATTern:ID:MODE command specifies whether a standard ID value or an extended ID value is used when searching for a CAN event.

The ID value is specified using the :SEARch:SERial:CAN:PATTern:ID command.

Query Syntax

```
:SEARch:SERial:CAN:PATTern:ID:MODE?
```

The :SEARch:SERial:CAN:PATTern:ID:MODE? query returns the current setting.

Return Format

```
<value><NL>

<value> ::= {STAN | EXT}
```

See Also

- Chapter 31, ":SEARch Commands," starting on page 821
- ":SEARch:SERial:CAN:MODE" on page 848
- ":SEARch:SERial:CAN:PATTern:ID" on page 852

:SEARch:SERial:CAN:SYMBolic:MESSage

(see page 1292)

Command Syntax

```
:SEARch:SERial:CAN:SYMBolic:MESSage <name>

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

The :SEARch:SERial:CAN:SYMBolic:MESSage command specifies the message to search for when CAN symbolic data has been loaded (recalled) into the oscilloscope and the CAN serial search mode is set to MESSage or MSIGnal.

Query Syntax

```
:SEARch:SERial:CAN:SYMBolic:MESSage?
```

The :SEARch:SERial:CAN:SYMBolic:MESSage? query returns the specified message.

Return Format

```
<name><NL>

<name> ::= quotes ASCII string
```

See Also

- ":RECall:DBC[:STARt]" on page 679
- ":SEARch:SERial:CAN:MODE" on page 848
- ":SEARch:SERial:CAN:SYMBolic:SIGNal" on page 855
- ":SEARch:SERial:CAN:SYMBolic:VALue" on page 856

:SEARch:SERial:CAN:SYMBolic:SIGNal

N (see page 1292)

Command Syntax

```
:SEARch:SERial:CAN:SYMBolic:SIGNal <name>

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

The :SEARch:SERial:CAN:SYMBolic:SIGNal command specifies the signal to search for when CAN symbolic data has been loaded (recalled) into the oscilloscope and the CAN serial search mode is set to MSIGnal.

Query Syntax

```
:SEARch:SERial:CAN:SYMBolic:SIGNal?
```

The :SEARch:SERial:CAN:SYMBolic:SIGNal? query returns the specified signal.

Return Format

```
<name><NL>

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

See Also

- ":RECall:DBC[:STARt]" on page 679
- ":SEARch:SERial:CAN:MODE" on page 848
- ":SEARch:SERial:CAN:SYMBolic:MESSage" on page 854
- ":SEARch:SERial:CAN:SYMBolic:VALue" on page 856

:SEARch:SERial:CAN:SYMBolic:VALue

(see page 1292)

Command Syntax

```
:SEARch:SERial:CAN:SYMBolic:VALue <data>

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

The :SEARch:SERial:CAN:SYMBolic:VALue command specifies the signal value to search for when CAN symbolic data has been loaded (recalled) into the oscilloscope and the CAN serial search mode is set to MSIGnal.

NOTE Encoded signal values are not supported in the remote interface (even though they can be used in the front panel graphical interface).

Query Syntax

```
:SEARch:SERial:CAN:SYMBolic:VALue?
```

The :SEARch:SERial:CAN:SYMBolic:VALue? query returns the specified signal value.

Return Format

```
<data><NL>

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

See Also

- ":RECall:DBC[:STARt]" on page 679
- ":SEARch:SERial:CAN:MODE" on page 848
- ":SEARch:SERial:CAN:SYMBolic:MESSage" on page 854
- ":SEARch:SERial:CAN:SYMBolic:SIGNal" on page 855

:SEARch:SERial:IIC Commands

Table 107 :SEARch:SERial:IIC Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:IIC:MODE <value> (see page 858)	:SEARch:SERial:IIC:MODE? (see page 858)	<value> ::= {RESTart | ADDRess | ANACk | NACKnowledge | REPRom | READ7 | WRITE7 | R7Data2 | W7Data2}
:SEARch:SERial:IIC:PATTern:ADDRess <value> (see page 860)	:SEARch:SERial:IIC:PATTern:ADDRess? (see page 860)	<value> ::= integer
:SEARch:SERial:IIC:PATTern:DATA <value> (see page 861)	:SEARch:SERial:IIC:PATTern:DATA? (see page 861)	<value> ::= integer
:SEARch:SERial:IIC:PATTern:DATA2 <value> (see page 862)	:SEARch:SERial:IIC:PATTern:DATA2? (see page 862)	<value> ::= integer
:SEARch:SERial:IIC:QUALifier <value> (see page 863)	:SEARch:SERial:IIC:QUALifier? (see page 863)	<value> ::= {EQUal | NOTequal | LESSthan | GREaterthan}

:SEARch:SERial:IIC:MODE

N (see page 1292)

Command Syntax

```
:SEARch:SERial:IIC:MODE <value>

<value> ::= {RESTart | ADDRess | ANACk | NACKnowledge | REPRom
   | READ7 | WRITE7 | R7Data2 | W7Data2}
```

The :SEARch:SERial:IIC:MODE command selects the type of IIC information to find in the Lister display:

- RESTart – searches for another start condition occurring before a stop condition.
- ADDRess – searches for a packet with the specified address, ignoring the R/W bit.
- ANACk – searches for address with no acknowledge events.
- NACKnowledge – searches for missing acknowledge events.
- REPRom – searches for EEPROM data reads.
- READ7 – searches for 7-bit address frames containing Start:Address7:Read:Ack:Data. The value READ is also accepted for READ7.
- WRITe7 – searches for 7-bit address frames containing Start:Address7:Write:Ack:Data. The value WRITe is also accepted for WRITe7.
- R7Data2 – searches for 7-bit address frames containing Start:Address7:Read:Ack:Data:Ack:Data2.
- W7Data2 – searches for 7-bit address frames containing Start:Address7:Write:Ack:Data:Ack:Data2.

NOTE **The short form of READ7 (READ7), REPRom (REPR), and WRITe7 (WRIT7) do not follow the defined Long Form to Short Form Truncation Rules (see page 1294).**

When searching for events containing addresses, address values are specified using the :SEARch:SERial:IIC:PATTern:ADDRess command.

When searching for events containing data, data values are specified using the :SEARch:SERial:IIC:PATTern:DATA and :SEARch:SERial:IIC:PATTern:DATA2 commands.

Query Syntax

```
:SEARch:SERial:IIC:MODE?
```

The :SEARch:SERial:IIC:MODE? query returns the currently selected mode.

Return Format

```
<value><NL>

<value> ::= {REST | ADDR | ANAC | NACK | REPR | READ7 | WRITE7
   | R7D2 | W7D2}
```

See Also

- Chapter 31, ":SEARch Commands," starting on page 821
- ":SEARch:SERial:IIC:PATTern:ADDRess" on page 860
- ":SEARch:SERial:IIC:PATTern:DATA" on page 861
- ":SEARch:SERial:IIC:PATTern:DATA2" on page 862
- ":SEARch:SERial:IIC:QUALifier" on page 863

:SEARch:SERial:IIC:PATTern:ADDRess

(see page 1292)

Command Syntax

```
:SEARch:SERial:IIC:PATTern:ADDRess <value>

<value> ::= integer
```

The :SEARch:SERial:IIC:PATTern:ADDRess command specifies address values when searching for IIC events.

To set don't care values, use the integer -1.

Query Syntax

```
:SEARch:SERial:IIC:PATTern:ADDRess?
```

The :SEARch:SERial:IIC:PATTern:ADDRess? query returns the current address value setting.

Return Format

```
<value><NL>

<value> ::= integer
```

See Also

- Chapter 31, ":SEARch Commands," starting on page 821
- ":SEARch:SERial:IIC:MODE" on page 858

:SEARch:SERial:IIC:PATTern:DATA

 (see page 1292)

Command Syntax

```
:SEARch:SERial:IIC:PATTern:DATA <value>

<value> ::= integer
```

The :SEARch:SERial:IIC:PATTern:DATA command specifies data values when searching for IIC events.

To set don't care values, use the integer -1.

When searching for IIC EEPROM data read events, you specify the data value qualifier using the :SEARch:SERial:IIC:QUALifier command.

Query Syntax

```
:SEARch:SERial:IIC:PATTern:DATA?
```

The :SEARch:SERial:IIC:PATTern:DATA? query returns the current data value setting.

Return Format

```
<value><NL>

<value> ::= integer
```

See Also

- Chapter 31, ":SEARch Commands," starting on page 821
- ":SEARch:SERial:IIC:MODE" on page 858
- ":SEARch:SERial:IIC:QUALifier" on page 863
- ":SEARch:SERial:IIC:PATTern:DATA2" on page 862

:SEARch:SERial:IIC:PATTern:DATA2

(see page 1292)

Command Syntax

```
:SEARch:SERial:IIC:PATTern:DATA2 <value>

<value> ::= integer
```

The :SEARch:SERial:IIC:PATTern:DATA2 command specifies the second data value when searching for IIC events with two data values.

To set don't care values, use the integer -1.

Query Syntax

```
:SEARch:SERial:IIC:PATTern:DATA2?
```

The :SEARch:SERial:IIC:PATTern:DATA2? query returns the current second data value setting.

Return Format

```
<value><NL>

<value> ::= integer
```

See Also

- Chapter 31, ":SEARch Commands," starting on page 821
- ":SEARch:SERial:IIC:MODE" on page 858
- ":SEARch:SERial:IIC:PATTern:DATA" on page 861

:SEARch:SERial:IIC:QUALifier

(see page 1292)

Command Syntax

```
:SEARch:SERial:IIC:QUALifier <value>

<value> ::= {EQUal | NOTequal | LESSthan | GREaterthan}
```

The :SEARch:SERial:IIC:QUALifier command specifies the data value qualifier used when searching for IIC EEPROM data read events.

Query Syntax

```
:SEARch:SERial:IIC:QUALifier?
```

The :SEARch:SERial:IIC:QUALifier? query returns the current data value qualifier setting.

Return Format

```
<value><NL>

<value> ::= {EQU | NOT | LESS | GRE}
```

See Also

- Chapter 31, ":SEARch Commands," starting on page 821
- ":SEARch:SERial:IIC:MODE" on page 858
- ":SEARch:SERial:IIC:PATTern:DATA" on page 861

:SEARch:SERial:LIN Commands

Table 108 :SEARch:SERial:LIN Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:LIN:ID <value> (see page 865)	:SEARch:SERial:LIN:ID? (see page 865)	<value> ::= 7-bit integer in decimal or <nondecimal> from 0-63 <nondecimal> ::= #Hnn where n ::= {0,..,9 | A,..,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary
:SEARch:SERial:LIN:MODE <value> (see page 866)	:SEARch:SERial:LIN:MODE? (see page 866)	<value> ::= {ID | DATA | ERRor}
:SEARch:SERial:LIN:PATTern:DATA <string> (see page 867)	:SEARch:SERial:LIN:PATTern:DATA? (see page 867)	When :SEARch:SERial:LIN:PATTern:FORMat DECimal, <string> ::= "n" where n ::= 32-bit integer in unsigned decimal, returns "$" if data has any don't cares When :SEARch:SERial:LIN:PATTern:FORMat HEX, <string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X}
:SEARch:SERial:LIN:PATTern:DATA:LENGth <length> (see page 868)	:SEARch:SERial:LIN:PATTern:DATA:LENGth? (see page 868)	<length> ::= integer from 1 to 8 in NR1 format
:SEARch:SERial:LIN:PATTern:FORMat <base> (see page 869)	:SEARch:SERial:LIN:PATTern:FORMat? (see page 869)	<base> ::= {HEX | DECimal}
:SEARch:SERial:LIN:SYMBolic:FRAMe <name> (see page 870)	:SEARch:SERial:LIN:SYMBolic:FRAMe? (see page 870)	<name> ::= quoted ASCII string
:SEARch:SERial:LIN:SYMBolic:SIGNal <name> (see page 871)	:SEARch:SERial:LIN:SYMBolic:SIGNal? (see page 871)	<name> ::= quoted ASCII string
:SEARch:SERial:LIN:SYMBolic:VALue <data> (see page 872)	:SEARch:SERial:LIN:SYMBolic:VALue? (see page 872)	<data> ::= value in NR3 format

:SEARch:SERial:LIN:ID

(see page 1292)

Command Syntax

```
:SEARch:SERial:LIN:ID <value>

<value> ::= 7-bit integer in decimal or <nondecimal> from 0-63

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

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

The :SEARch:SERial:LIN:ID command specifies the frame ID value when searching for LIN events.

Query Syntax

```
:SEARch:SERial:LIN:ID?
```

The :SEARch:SERial:LIN:ID? query returns the current frame ID setting.

Return Format

```
<value><NL>

<value> ::= 7-bit integer in decimal
```

See Also

- Chapter 31, ":SEARch Commands," starting on page 821
- ":SEARch:SERial:LIN:MODE" on page 866

:SEARch:SERial:LIN:MODE

 (see page 1292)

Command Syntax

```
:SEARch:SERial:LIN:MODE <value>

<value> ::= {ID | DATA | ERRor | FRAMe | FSIGnal}
```

The :SEARch:SERial:LIN:MODE command selects the type of LIN information to find in the Lister display:

- ID – searches for a frame ID.
- DATA – searches for a frame ID and data.
- ERRor – searches for errors.
- FRAMe – searches for symbolic frames.
- FSIGnal – searched for symbolic frames and a signal values.

Frame IDs are specified using the :SEARch:SERial:LIN:ID command.

Data values are specified using the :SEARch:SERial:LIN:PATTern:DATA command.

Frames, signals, and signal values are specified using the :SEARch:SERial:LIN:SYMBolic:FRAMe, :SEARch:SERial:LIN:SYMBolic:SIGNal, and :SEARch:SERial:LIN:SYMBolic:VALue commands. LIN symbolic data files are loaded (recalled) using the :RECall:LDF[:STARt] command.

Query Syntax

```
:SEARch:SERial:LIN:MODE?
```

The :SEARch:SERial:LIN:MODE? query returns the currently selected mode.

Return Format

```
<value><NL>

<value> ::= {ID | DATA | ERR | FRAM | FSIG}
```

See Also

- Chapter 31, ":SEARch Commands," starting on page 821
- ":SEARch:SERial:LIN:ID" on page 865
- ":SEARch:SERial:LIN:PATTern:DATA" on page 867
- ":RECall:LDF[:STARt]" on page 681
- ":SEARch:SERial:LIN:SYMBolic:FRAMe" on page 870
- ":SEARch:SERial:LIN:SYMBolic:SIGNal" on page 871
- ":SEARch:SERial:LIN:SYMBolic:VALue" on page 872

:SEARch:SERial:LIN:PATTern:DATA

N (see page 1292)

Command Syntax

```
:SEARch:SERial:LIN:PATTern:DATA <string>

When :SEARch:SERial:LIN:PATTern:FORMat DECimal,
<string> ::= "n" where n ::= 32-bit integer in unsigned decimal

When :SEARch:SERial:LIN:PATTern:FORMat HEX,
<string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X }
```

The :SEARch:SERial:LIN:PATTern:DATA command specifies the data value when searching for LIN events.

The number base of the value entered with this command is specified using the :SEARch:SERial:LIN:PATTern:FORMat command. To set don't care values with the DATA command, the FORMat must be HEX.

The length of the data value entered is specified using the :SEARch:SERial:LIN:PATTern:DATA:LENGth command.

Query Syntax

```
:SEARch:SERial:LIN:PATTern:DATA?
```

The :SEARch:SERial:LIN:PATTern:DATA? query returns the current data value setting.

Return Format

```
<string><NL>

When :SEARch:SERial:LIN:PATTern:FORMat DECimal,
<string> ::= "n" where n ::= 32-bit integer in unsigned decimal or
             "$" if data has any don't cares

When :SEARch:SERial:LIN:PATTern:FORMat HEX,
<string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X }
```

See Also

- Chapter 31, ":SEARch Commands," starting on page 821
- ":SEARch:SERial:LIN:MODE" on page 866
- ":SEARch:SERial:LIN:PATTern:FORMat" on page 869
- ":SEARch:SERial:LIN:PATTern:DATA:LENGth" on page 868

:SEARch:SERial:LIN:PATTern:DATA:LENGth

 (see page 1292)

Command Syntax

```
:SEARch:SERial:LIN:PATTern:DATA:LENGth <length>

<length> ::= integer from 1 to 8 in NR1 format
```

The :SEARch:SERial:LIN:PATTern:DATA:LENGth command specifies the the length of the data value when searching for LIN events.

The data value is specified using the :SEARch:SERial:LIN:PATTern:DATA command.

Query Syntax

```
:SEARch:SERial:LIN:PATTern:DATA:LENGth?
```

The :SEARch:SERial:LIN:PATTern:DATA:LENGth? query returns the current data value length setting.

Return Format

```
<length><NL>

<length> ::= integer from 1 to 8 in NR1 format
```

See Also

- Chapter 31, ":SEARch Commands," starting on page 821
- ":SEARch:SERial:LIN:PATTern:DATA" on page 867

:SEARch:SERial:LIN:PATTern:FORMat

(see page 1292)

Command Syntax

```
:SEARch:SERial:LIN:PATTern:FORMat <base>

<base> ::= {HEX | DECimal}
```

The :SEARch:SERial:LIN:PATTern:FORMat command specifies the number base used with the :SEARch:SERial:LIN:PATTern:DATA command.

Query Syntax

```
:SEARch:SERial:LIN:PATTern:FORMat?
```

The :SEARch:SERial:LIN:PATTern:FORMat? query returns the current number base setting.

Return Format

```
<base><NL>

<base> ::= {HEX | DEC}
```

See Also

- Chapter 31, ":SEARch Commands," starting on page 821
- ":SEARch:SERial:LIN:PATTern:DATA" on page 867

:SEARch:SERial:LIN:SYMBolic:FRAMe

(see page 1292)

Command Syntax

```
:SEARch:SERial:LIN:SYMBolic:FRAMe <name>

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

The :SEARch:SERial:LIN:SYMBolic:FRAMe command specifies the message to search for when LIN symbolic data has been loaded (recalled) into the oscilloscope and the LIN serial search mode is set to FRAMe or FSIGnal.

Query Syntax

```
:SEARch:SERial:LIN:SYMBolic:FRAMe?
```

The :SEARch:SERial:LIN:SYMBolic:FRAMe? query returns the specified message.

Return Format

```
<name><NL>

<name> ::= quotes ASCII string
```

See Also

- ":RECall:LDF[:STARt]" on page 681
- ":SEARch:SERial:LIN:MODE" on page 866
- ":SEARch:SERial:LIN:SYMBolic:SIGNal" on page 871
- ":SEARch:SERial:LIN:SYMBolic:VALue" on page 872

:SEARch:SERial:LIN:SYMBolic:SIGNal

 (see page 1292)

Command Syntax

```
:SEARch:SERial:LIN:SYMBolic:SIGNal <name>

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

The :SEARch:SERial:LIN:SYMBolic:SIGNal command specifies the signal to search for when LIN symbolic data has been loaded (recalled) into the oscilloscope and the LIN serial search mode is set to FSIGnal.

Query Syntax

```
:SEARch:SERial:LIN:SYMBolic:SIGNal?
```

The :SEARch:SERial:LIN:SYMBolic:SIGNal? query returns the specified signal.

Return Format

```
<name><NL>

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

See Also

- ":RECall:LDF[:STARt]" on page 681
- ":SEARch:SERial:LIN:MODE" on page 866
- ":SEARch:SERial:LIN:SYMBolic:FRAMe" on page 870
- ":SEARch:SERial:LIN:SYMBolic:VALue" on page 872

:SEARch:SERial:LIN:SYMBolic:VALue

(see page 1292)

Command Syntax

```
:SEARch:SERial:LIN:SYMBolic:VALue <data>

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

The :SEARch:SERial:LIN:SYMBolic:VALue command specifies the signal value to search for when LIN symbolic data has been loaded (recalled) into the oscilloscope and the LIN serial search mode is set to FSIGnal.

NOTE Encoded signal values are not supported in the remote interface (even though they can be used in the front panel graphical interface).

Query Syntax

```
:SEARch:SERial:LIN:SYMBolic:VALue?
```

The :SEARch:SERial:LIN:SYMBolic:VALue? query returns the specified signal value.

Return Format

```
<data><NL>

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

See Also

- ":RECall:LDF[:STARt]" on page 681
- ":SEARch:SERial:LIN:MODE" on page 866
- ":SEARch:SERial:LIN:SYMBolic:FRAMe" on page 870
- ":SEARch:SERial:LIN:SYMBolic:SIGNal" on page 871

:SEARch:SERial:SPI Commands

Table 109 :SEARch:SERial:SPI Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:SPI:MODE <value> (see page 874)	:SEARch:SERial:SPI:MODE? (see page 874)	<value> ::= {MOSI | MISO}
:SEARch:SERial:SPI:PATTern:DATA <string> (see page 875)	:SEARch:SERial:SPI:PATTern:DATA? (see page 875)	<string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X}
:SEARch:SERial:SPI:PATTern:WIDTh <width> (see page 876)	:SEARch:SERial:SPI:PATTern:WIDTh? (see page 876)	<width> ::= integer from 1 to 10

:SEARch:SERial:SPI:MODE

 (see page 1292)

Command Syntax

```
:SEARch:SERial:SPI:MODE <value>

<value> ::= {MOSI | MISO}
```

The :SEARch:SERial:SPI:MODE command specifies whether the SPI search will be on the MOSI data or the MISO data.

Data values are specified using the :SEARch:SERial:SPI:PATTern:DATA command.

Query Syntax

```
:SEARch:SERial:SPI:MODE?
```

The :SEARch:SERial:SPI:MODE? query returns the current SPI search mode setting.

Return Format

```
<value><NL>

<value> ::= {MOSI | MISO}
```

See Also

- Chapter 31, ":SEARch Commands," starting on page 821
- ":SEARch:SERial:SPI:PATTern:DATA" on page 875

:SEARch:SERial:SPI:PATTern:DATA

N (see page 1292)

Command Syntax

```
:SEARch:SERial:SPI:PATTern:DATA <string>

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

The :SEARch:SERial:SPI:PATTern:DATA command specifies the data value when searching for SPI events.

The width of the data value is specified using the :SEARch:SERial:SPI:PATTern:WIDTh command.

Query Syntax

```
:SEARch:SERial:SPI:PATTern:DATA?
```

The :SEARch:SERial:SPI:PATTern:DATA? query returns the current data value setting.

Return Format

```
<string><NL>

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

See Also

- Chapter 31, ":SEARch Commands," starting on page 821
- ":SEARch:SERial:SPI:PATTern:WIDTh" on page 876

:SEARch:SERial:SPI:PATTern:WIDTh

 (see page 1292)

Command Syntax

```
:SEARch:SERial:SPI:PATTern:WIDTh <width>

<width> ::= integer from 1 to 10
```

The :SEARch:SERial:SPI:PATTern:WIDTh command specifies the width of the data value (in bytes) when searching for SPI events.

The data value is specified using the :SEARch:SERial:SPI:PATTern:DATA command.

Query Syntax

```
:SEARch:SERial:SPI:PATTern:WIDTh?
```

The :SEARch:SERial:SPI:PATTern:WIDTh? query returns the current data width setting.

Return Format

```
<width><NL>

<width> ::= integer from 1 to 10
```

See Also

- Chapter 31, ":SEARch Commands," starting on page 821
- ":SEARch:SERial:SPI:PATTern:DATA" on page 875

:SEARch:SERial:UART Commands

Table 110 :SEARch:SERial:UART Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:UART:DATA <value> (see page 878)	:SEARch:SERial:UART:DATA? (see page 878)	<value> ::= 8-bit integer from 0-255 (0x00-0xff) in decimal, <hexadecimal>, or <binary> format <hexadecimal> ::= #Hnn where n ::= {0,..,9| A,..,F} for hexadecimal <binary> ::= #Bnn...n where n ::= {0 | 1} for binary
:SEARch:SERial:UART:MODE <value> (see page 879)	:SEARch:SERial:UART:MODE? (see page 879)	<value> ::= {RDATa | RD1 | RD0 | RDX | TDATa | TD1 | TD0 | TDX | RXPArity | TXPArity | AERRor}
:SEARch:SERial:UART:QUALifier <value> (see page 881)	:SEARch:SERial:UART:QUALifier? (see page 881)	<value> ::= {EQUal | NOTequal | GREaterthan | LESSthan}

:SEARch:SERial:UART:DATA

(see page 1292)

Command Syntax

```
:SEARch:SERial:UART:DATA <value>

<value> ::= 8-bit integer from 0-255 (0x00-0xff) in decimal,
            <hexadecimal>, or <binary> format

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

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

The :SEARch:SERial:UART:DATA command specifies a data value when searching for UART/RS232 events.

The data value qualifier is specified using the :SEARch:SERial:UART:QUALifier command.

Query Syntax

```
:SEARch:SERial:UART:DATA?
```

The :SEARch:SERial:UART:DATA? query returns the current data value setting.

Return Format

```
<value><NL>

<value> ::= 8-bit integer from 0-255 (0x00-0xff) in decimal format
```

See Also

- Chapter 31, ":SEARch Commands," starting on page 821
- ":SEARch:SERial:UART:MODE" on page 879
- ":SEARch:SERial:UART:QUALifier" on page 881

:SEARch:SERial:UART:MODE

(see page 1292)

Command Syntax

```
:SEARch:SERial:UART:MODE <value>

<value> ::= {RDATa | RD1 | RD0 | RDX | TDATa | TD1 | TD0 | TDX
   | RXPArity | TXPArity | AERRor}
```

The :SEARch:SERial:UART:MODE command selects the type of UART/RS232 information to find in the Lister display:

- RDATa – searches for a receive data value when data words are from 5 to 8 bits long.
- RD1 – searches for a receive data value when data words are 9 bits long and the 9th (alert) bit is 1.
- RD0 – searches for a receive data value when data words are 9 bits long and the 9th (alert) bit is 0.
- RDX – searches for a receive data value when data words are 9 bits long and the 9th (alert) bit is a don't care (X).
- TDATa – searches for a transmit data value when data words are from 5 to 8 bits long.
- TD1 – searches for a transmit data value when data words are 9 bits long and the 9th (alert) bit is 1.
- TD0 – searches for a transmit data value when data words are 9 bits long and the 9th (alert) bit is 0.
- TDX – searches for a transmit data value when data words are 9 bits long and the 9th (alert) bit is a don't care (X).
- RXPArity – searches for receive data parity errors.
- TXPArity – searches for transmit data parity errors.
- AERRor – searches for any error.

Data values are specified using the :SEARch:SERial:UART:DATA command.

Data value qualifiers are specified using the :SEARch:SERial:UART:QUALifier command.

Query Syntax

```
:SEARch:SERial:UART:MODE?
```

The :SEARch:SERial:UART:MODE? query returns ...

Return Format

```
<value><NL>

<value> ::= {RDAT | RD1 | RD0 | RDX | TDAT | TD1 | TD0 | TDX | RXPA
   | TXPA | AERR}
```

See Also

- Chapter 31, ":SEARch Commands," starting on page 821
- ":SEARch:SERial:UART:DATA" on page 878

- ":SEARch:SERial:UART:QUALifier" on page 881

:SEARch:SERial:UART:QUALifier

 (see page 1292)

Command Syntax

```
:SEARch:SERial:UART:QUALifier <value>

<value> ::= {EQUal | NOTequal | GREaterthan | LESSthan}
```

The :SEARch:SERial:UART:QUALifier command specifies the data value qualifier when searching for UART/RS232 events.

Query Syntax

```
:SEARch:SERial:UART:QUALifier?
```

The :SEARch:SERial:UART:QUALifier? query returns the current data value qualifier setting.

Return Format

```
<value><NL>

<value> ::= {EQU | NOT | GRE | LESS}
```

See Also

- Chapter 31, ":SEARch Commands," starting on page 821
- ":SEARch:SERial:UART:DATA" on page 878

32 :STATus Commands

The STATus commands subsystem controls the status registers defined for the HD3-Series oscilloscopes. See:

Introduction to :STATus Commands

The commands in the STATus commands subsystem control the status registers defined for the HD3-Series oscilloscopes.

The SCPI-standard Standard Event Status Register, Status Byte Register, and Service Request Enable Register are also present in the HD3-Series oscilloscopes, but they are controlled with the common (*) commands: *ESR?, *ESE, *STB?, and *SRE.

The status registers defined for the HD3-Series oscilloscopes follow the SCPI status register model where a Condition register continuously monitors status, Transition Filters allow status transitions to be latched into an Event register, and an Enable register allows events to affect a Summary Message that is sent to a parent in the status register hierarchy.

Every status register defined for the HD3-Series oscilloscopes is actually a collection of three registers and two transition filters that let you manage particular status events.

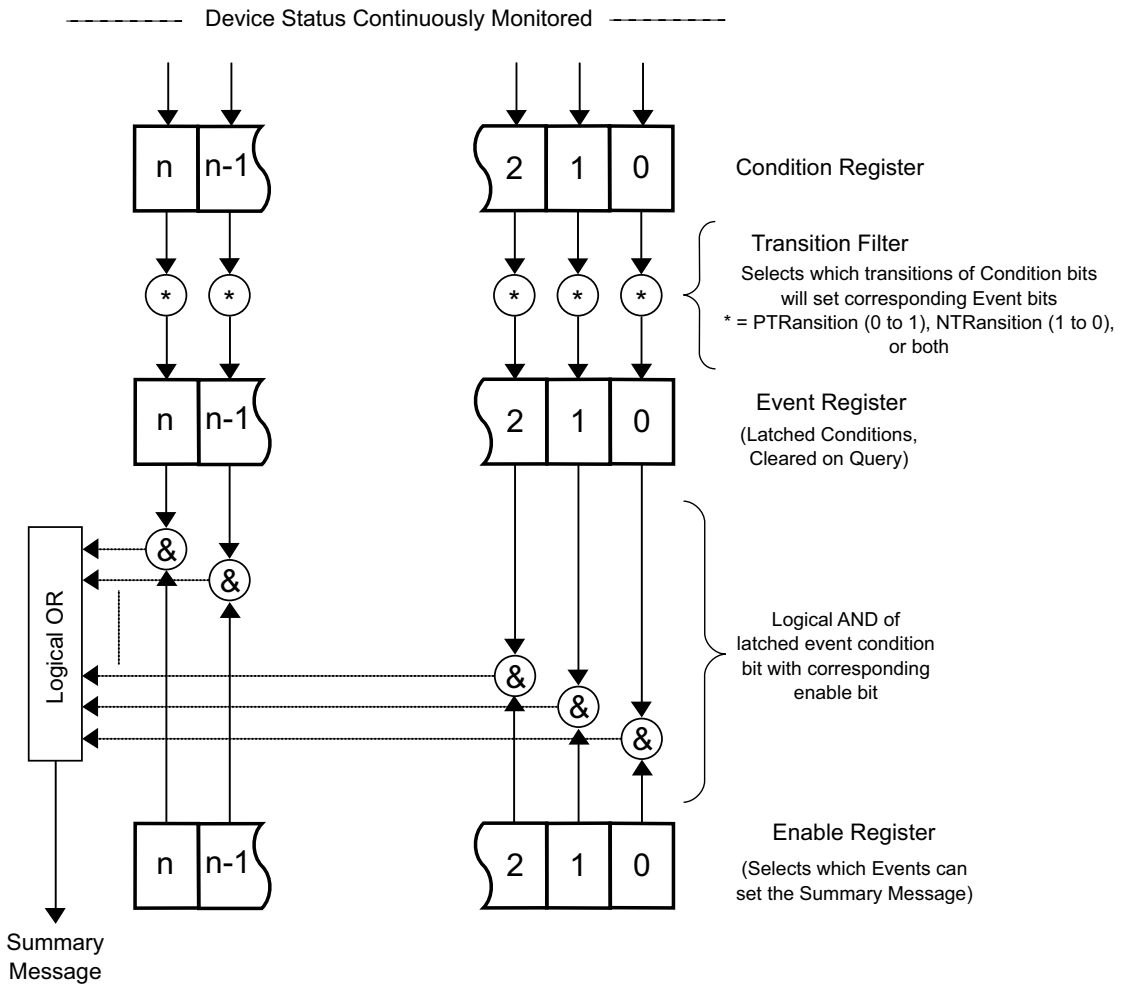


Figure 3 SCPI Status Register Model

So, for each status register defined for the HD3-Series oscilloscopes, there are these commands and queries:

- :STATus:<register>:CONDition? – For reading the Condition register.
- :STATus:<register>:PTRansition – For specifying the bits on which a positive transition (0 to 1 or FALSE to TRUE) will be latched into the Event register.
- :STATus:<register>:NTRansition – For specifying the bits on which a negative transition (1 to 0 or TRUE to FALSE) will be latched into the Event register.

  Both positive and negative transitions can be enabled for a bit. In this case, any transition is latched into the Event register.
- :STATus:<register>[:EVENt]? – For reading whether events have occured since the last time the register was read.

Reading an Event register clears it.

- :STATus:<register>:ENABle — For specifying which bits affect the Summary Message.

And there are these variations for reading and setting individual bits:

- :STATus:<register>:BIT<b>:CONDition?
- :STATus:<register>:BIT<b>:PTRansition
- :STATus:<register>:BIT<b>:NTRansition
- :STATus:<register>:BIT<b>[:EVENt]?

  Reading an individual bit in the Event register clears only that bit.
- :STATus:<register>:BIT<b>:ENABle

The status registers defined for the HD3-Series oscilloscopes are 16-bit registers, but the most-significant bit (bit 15) is never used and always returns zero.

See Also

- "*ESR? (Standard Event Status Register)" on page 187
- "*ESE (Standard Event Status Enable)" on page 185
- "*STB? (Read Status Byte)" on page 200
- "*SRE (Service Request Enable)" on page 198

General :STATus Commands

Table 111 General :STATus Commands Summary

Command	Query	Options and Query Returns
`:STATus:PRESet` (see page 887)	`n/a`	`n/a`

:STATus:PRESet

 (see page 1292)

Command Syntax `:STATus:PRESet`

The :STATus:PRESet command sets all the transition filters and enable registers to their default state.

See Also

- "Introduction to :STATus Commands" on page 883
- "General :STATus Commands" on page 886
- ":STATus:OPERation Commands" on page 888
- ":STATus:OPERation:ARM Commands" on page 903
- ":STATus:OPERation:HARDware Commands" on page 916
- ":STATus:OPERation:LOCal Commands" on page 929
- ":STATus:OPERation:MTESt Commands" on page 942
- ":STATus:OPERation:OVERload Commands" on page 955
- ":STATus:OPERation:OVERload:PFAult Commands" on page 968
- ":STATus:OPERation:POWer Commands" on page 981
- ":STATus:TRIGger Commands" on page 994
- ":STATus:USER Commands" on page 1007
- "*CLS (Clear Status)" on page 184

:STATus:OPERation Commands

The :STATus:OPERation commands control the Operation Status Register.

Table 112 :STATus:OPERation Commands Summary

Command	Query	Options and Query Returns
n/a	:STATus:OPERation:BIT <b>:CONDition? (see page 893)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:BIT <b>:ENABle <bit_value> (see page 894)	:STATus:OPERation:BIT <b>:ENABle? (see page 894)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:BIT <b>:NTRansition <bit_value> (see page 895)	:STATus:OPERation:BIT <b>:NTRansition? (see page 895)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:BIT <b>:PTRansition <bit_value> (see page 896)	:STATus:OPERation:BIT <b>:PTRansition? (see page 896)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation:BIT <b>[:EVENt]? (see page 897)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation:CONDition? (see page 898)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:ENABle <sum_of_bits> (see page 899)	:STATus:OPERation:ENABle? (see page 899)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:NTRansition <sum_of_bits> (see page 900)	:STATus:OPERation:NTRansition? (see page 900)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format

Table 112 :STATus:OPERation Commands Summary (continued)

Command	Query	Options and Query Returns
:STATus:OPERation:PTRansition <sum_of_bits> (see page 901)	:STATus:OPERation:PTRansition? (see page 901)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation[:EVENt]? (see page 902)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format

Operation Status Register

To see where the Operation Status Register appears in the status register hierarchy, see Chapter 41, “Status Reporting,” starting on page 1253.

To unserstand the SCPI status register model, see "Introduction to :STATus Commands" on page 883.

The following bits appear in the Operation Status Register.

Table 113 Operation Status Register Bits

Bit	Name	Description	When Set (1 = High = True), Indicates:	From:
15	---	---	(Not used, always zero.)	---
14	IOF	IO Operation Failed	Event only when something causes the IO operation to fail, like disconnecting a USB device or interrupting the operation from the oscilloscope's front panel.	---
13	IOC	IO Operation Complete	Event when any IO operation completes. IO operations are any remote data request using any interface (USB or LAN). For example, if you connect to an oscilloscope using the USB interface and then request waveform data, the IOC bit will be set when the IO operation completes.	---
12	HWE	Hardware Event	Event when hardware event occurs.	Hardware Operation Register
11	OVLR	Overload	A 50Ω input overload has occurred.	Overload Operation Register
10	HIST	Histogram	A histogram counter overflow has occurred.	---
9	MTE	Mask Test Event	A mask test event has occurred.	Mask Test Operation Register
8	LCL	Local	A Test Fail or Test Complete has occurred.	Local Operation Register
7	Powe r	Power	Can potentially indicate a power measurements application Deskew, Apply, or Setup is complete,	Power Operation Register
6	---	---	(Not used.)	---
5	Wait Trig	Wait Trig	The trigger is armed (set by the Trigger Armed Event Register (TER)).	Arm Event Register (can also use :AER?)

Table 113 Operation Status Register Bits (continued)

Bit	Name	Description	When Set (1 = High = True), Indicates:	From:
4	RUI Enab	Remote Enabled	The remote user interface has gone from a disabled state to an enabled state. The front-panel graphical user interface can disable most of the remote interface, including the *OPC syntax, for example when: ▪ a modal dialog box is open ▪ the user is being prompted ▪ all segments are being analyzed ▪ there is a channel overload ▪ certain compliance applications are running When disabled, commands or queries are accepted, but "settings conflict" errors are returned. The status model is the only part of the remote user interface that is enabled. To determine when the remote interface is re-enabled, you can read this bit or wait for the event that gets generated when its status goes from disabled to enabled.	---
3	Run	Running	The oscilloscope has gone from a stop state to a single or running state.	---
2	---	---	(Not used.)	---
1	---	---	(Not used.)	---
0	Calibrating	Calibrating	The oscilloscope is calibrating.	---

When enabled, the summary message for the Operation Status Register goes to Status Byte Register bit 7 (OPER). See "*CLS (Clear Status)" on page 184.

:STATus:OPERation:BIT<b>:CONDition?

(see page 1292)

Query Syntax

```
:STATus:OPERation:BIT<b>:CONDition?

<b> ::= 0-14; an integer in NR1 format
```

The :STATus:OPERation:BIT<b>:CONDition? query returns the value of a particular bit in the Condition register.

The Condition register continuously monitors status. Reading a Condition register bit does not affect its contents.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:OPERation:BIT<b>:ENABle" on page 894
- ":STATus:OPERation:BIT<b>:NTRansition" on page 895
- ":STATus:OPERation:BIT<b>:PTRansition" on page 896
- ":STATus:OPERation:BIT<b>[:EVENt]?" on page 897
- ":STATus:OPERation:CONDition?" on page 898
- ":STATus:OPERation:ENABle" on page 899
- ":STATus:OPERation:NTRansition" on page 900
- ":STATus:OPERation:PTRansition" on page 901
- ":STATus:OPERation[:EVENt]?" on page 902
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:BIT<b>:ENABle

 (see page 1292)

Command Syntax

```
:STATus:OPERation:BIT<b>:ENABle <bit_value>

<b> ::= 0-14; an integer in NR1 format

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

The :STATus:OPERation:BIT<b>:ENABle command sets a particular bit in the Enable register that identifies a corresponding Event register bit that will be ORed with other identified Event register bits to create the Summary Message bit that is sent to the parent status register.

Query Syntax

```
:STATus:OPERation:BIT<b>:ENABle?
```

The :STATus:OPERation:BIT<b>:ENABle? query returns the value of the specified Enable register bit.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:OPERation:BIT<b>:CONDition?" on page 893
- ":STATus:OPERation:BIT<b>:NTRansition" on page 895
- ":STATus:OPERation:BIT<b>:PTRansition" on page 896
- ":STATus:OPERation:BIT<b>[:EVENt]?" on page 897
- ":STATus:OPERation:CONDition?" on page 898
- ":STATus:OPERation:ENABle" on page 899
- ":STATus:OPERation:NTRansition" on page 900
- ":STATus:OPERation:PTRansition" on page 901
- ":STATus:OPERation[:EVENt]?" on page 902
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:BIT<b>:NTRansition

N (see page 1292)

Command Syntax

```
:STATus:OPERation:BIT<b>:NTRansition <bit_value>

<b> ::= 0-14; an integer in NR1 format

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

The :STATus:OPERation:BIT<b>:NTRansition command sets a particular bit in the Negative Transition filter that identifies a Condition register bit on which a transition from 1 to 0 or TRUE to FALSE will be latched into the corresponding Event register bit.

Query Syntax

```
:STATus:OPERation:BIT<b>:NTRansition?
```

The :STATus:OPERation:BIT<b>:NTRansition? query returns the value of the specified Negative Transition filter bit.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:OPERation:BIT<b>:CONDition?" on page 893
- ":STATus:OPERation:BIT<b>:ENABle" on page 894
- ":STATus:OPERation:BIT<b>:PTRansition" on page 896
- ":STATus:OPERation:BIT<b>[:EVENt]?" on page 897
- ":STATus:OPERation:CONDition?" on page 898
- ":STATus:OPERation:ENABle" on page 899
- ":STATus:OPERation:NTRansition" on page 900
- ":STATus:OPERation:PTRansition" on page 901
- ":STATus:OPERation[:EVENt]?" on page 902
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:BIT<b>:PTRansition

(see page 1292)

Command Syntax

```
:STATus:OPERation:BIT<b>:PTRansition <bit_value>

<b> ::= 0-14; an integer in NR1 format

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

The :STATus:OPERation:BIT<b>:PTRansition command sets a particular bit in the Positive Transition filter that identifies a Condition register bit on which a transition from 0 to 1 or FALSE to TRUE will be latched into the corresponding Event register bit.

Query Syntax

```
:STATus:OPERation:BIT<b>:PTRansition?
```

The :STATus:OPERation:BIT<b>:PTRansition? query returns the value of the specified Positive Transition filter bit.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:OPERation:BIT<b>:CONDition?" on page 893
- ":STATus:OPERation:BIT<b>:ENABle" on page 894
- ":STATus:OPERation:BIT<b>:NTRansition" on page 895
- ":STATus:OPERation:BIT<b>[:EVENt]?" on page 897
- ":STATus:OPERation:CONDition?" on page 898
- ":STATus:OPERation:ENABle" on page 899
- ":STATus:OPERation:NTRansition" on page 900
- ":STATus:OPERation:PTRansition" on page 901
- ":STATus:OPERation[:EVENt]?" on page 902
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:BIT<b>[:EVENt]?

N (see page 1292)

Query Syntax

```
:STATus:OPERation:BIT<b>[:EVENt]?

<b> ::= 0-14; an integer in NR1 format
```

The :STATus:OPERation:BIT<b>[:EVENt]? query returns the value of the specified Event register bit and clears the bit.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:OPERation:BIT<b>:CONDition?" on page 893
- ":STATus:OPERation:BIT<b>:ENABle" on page 894
- ":STATus:OPERation:BIT<b>:NTRansition" on page 895
- ":STATus:OPERation:BIT<b>:PTRansition" on page 896
- ":STATus:OPERation:CONDition?" on page 898
- ":STATus:OPERation:ENABle" on page 899
- ":STATus:OPERation:NTRansition" on page 900
- ":STATus:OPERation:PTRansition" on page 901
- ":STATus:OPERation[:EVENt]?" on page 902
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:CONDition?

(see page 1292)

Query Syntax `:STATus:OPERation:CONDition?`

The :STATus:OPERation:CONDition? query returns the decimal value of the sum of the bits in the Condition register.

The Condition register continuously monitors status. Reading the Condition register does not affect its contents.

Return Format `<sum_of_bits><NL>`

`<sum_of_bits> ::= 0-32767; an integer in NR1 format`

See Also

- ":STATus:OPERation:BIT<b>:CONDition?" on page 893
- ":STATus:OPERation:BIT<b>:ENABle" on page 894
- ":STATus:OPERation:BIT<b>:NTRansition" on page 895
- ":STATus:OPERation:BIT<b>:PTRansition" on page 896
- ":STATus:OPERation:BIT<b>[:EVENt]?" on page 897
- ":STATus:OPERation:ENABle" on page 899
- ":STATus:OPERation:NTRansition" on page 900
- ":STATus:OPERation:PTRansition" on page 901
- ":STATus:OPERation[:EVENt]?" on page 902
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:ENABle

(see page 1292)

Command Syntax

```
:STATus:OPERation:ENABle <sum_of_bits>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

The :STATus:OPERation:ENABle command sets bits in the Enable register that identify which Event register bits are ORed to create the Summary Message bit that is sent to the parent status register.

Query Syntax

```
:STATus:OPERation:ENABle?
```

The :STATus:OPERation:ENABle? query returns the decimal value of the sum of the bits in the Enable register.

Return Format

```
<sum_of_bits><NL>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

See Also

- ":STATus:OPERation:BIT<b>:CONDition?" on page 893
- ":STATus:OPERation:BIT<b>:ENABle" on page 894
- ":STATus:OPERation:BIT<b>:NTRansition" on page 895
- ":STATus:OPERation:BIT<b>:PTRansition" on page 896
- ":STATus:OPERation:BIT<b>[:EVENt]?" on page 897
- ":STATus:OPERation:CONDition?" on page 898
- ":STATus:OPERation:NTRansition" on page 900
- ":STATus:OPERation:PTRansition" on page 901
- ":STATus:OPERation[:EVENt]?" on page 902
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:NTRansition

N (see page 1292)

Command Syntax

```
:STATus:OPERation:NTRansition <sum_of_bits>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

The :STATus:OPERation:NTRansition command sets bits in the Negative Transition filter that identify the Condition register bits on which transitions from 1 to 0 or TRUE to FALSE will be latched into the Event register.

Query Syntax

```
:STATus:OPERation:NTRansition?
```

The :STATus:OPERation:NTRansition? query returns the decimal value of the sum of the bits in the Negative Transition filter.

Return Format

```
<sum_of_bits><NL>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

See Also

- ":STATus:OPERation:BIT<b>:CONDition?" on page 893
- ":STATus:OPERation:BIT<b>:ENABle" on page 894
- ":STATus:OPERation:BIT<b>:NTRansition" on page 895
- ":STATus:OPERation:BIT<b>:PTRansition" on page 896
- ":STATus:OPERation:BIT<b>[:EVENt]?" on page 897
- ":STATus:OPERation:CONDition?" on page 898
- ":STATus:OPERation:ENABle" on page 899
- ":STATus:OPERation:PTRansition" on page 901
- ":STATus:OPERation[:EVENt]?" on page 902
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:PTRansition

 (see page 1292)

Command Syntax

```
:STATus:OPERation:PTRansition <sum_of_bits>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

The :STATus:OPERation:PTRansition command sets bits in the Positive Transition filter that identify the Condition register bits on which transitions from 0 to 1 or FALSE to TRUE will be latched into the Event register.

Query Syntax

```
:STATus:OPERation:PTRansition?
```

The :STATus:OPERation:PTRansition? query returns the decimal value of the sum of the bits in the Positive Transition filter.

Return Format

```
<sum_of_bits><NL>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

See Also

- ":STATus:OPERation:BIT<b>:CONDition?" on page 893
- ":STATus:OPERation:BIT<b>:ENABle" on page 894
- ":STATus:OPERation:BIT<b>:NTRansition" on page 895
- ":STATus:OPERation:BIT<b>:PTRansition" on page 896
- ":STATus:OPERation:BIT<b>[:EVENt]?" on page 897
- ":STATus:OPERation:CONDition?" on page 898
- ":STATus:OPERation:ENABle" on page 899
- ":STATus:OPERation:NTRansition" on page 900
- ":STATus:OPERation[:EVENt]?" on page 902
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation[:EVENt]?

(see page 1292)

Query Syntax `:STATus:OPERation[:EVENt]?`

The :STATus:OPERation[:EVENt]? query returns the decimal value of the sum of the bits in the Event register and clears the register.

The contents of the Event register show the events that have occurred since the last time the register was read.

Return Format `<sum_of_bits><NL>`

`<sum_of_bits> ::= 0-32767; an integer in NR1 format`

See Also

- ":STATus:OPERation:BIT<b>:CONDition?" on page 893
- ":STATus:OPERation:BIT<b>:ENABle" on page 894
- ":STATus:OPERation:BIT<b>:NTRansition" on page 895
- ":STATus:OPERation:BIT<b>:PTRansition" on page 896
- ":STATus:OPERation:BIT<b>[:EVENt]?" on page 897
- ":STATus:OPERation:CONDition?" on page 898
- ":STATus:OPERation:ENABle" on page 899
- ":STATus:OPERation:NTRansition" on page 900
- ":STATus:OPERation:PTRansition" on page 901
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:ARM Commands

The :STATus:OPERation:ARM commands control the Arm Event Register.

Table 114 :STATus:OPERation:ARM Commands Summary

Command	Query	Options and Query Returns
n/a	:STATus:OPERation:ARM :BIT<b>:CONDition? (see page 906)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:ARM :BIT<b>:ENABle <bit_value> (see page 907)	:STATus:OPERation:ARM :BIT<b>:ENABle? (see page 907)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:ARM :BIT<b>:NTRansition <bit_value> (see page 908)	:STATus:OPERation:ARM :BIT<b>:NTRansition? (see page 908)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:ARM :BIT<b>:PTRansition <bit_value> (see page 909)	:STATus:OPERation:ARM :BIT<b>:PTRansition? (see page 909)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation:ARM :BIT<b>[:EVENt]? (see page 910)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation:ARM :CONDition? (see page 911)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:ARM :ENABle <sum_of_bits> (see page 912)	:STATus:OPERation:ARM :ENABle? (see page 912)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:ARM :NTRansition <sum_of_bits> (see page 913)	:STATus:OPERation:ARM :NTRansition? (see page 913)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format

Table 114 :STATus:OPERation:ARM Commands Summary (continued)

Command	Query	Options and Query Returns
:STATus:OPERation:ARM :PTRansition <sum_of_bits> (see page 914)	:STATus:OPERation:ARM :PTRansition? (see page 914)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation:ARM [:EVENt]? (see page 915)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format

Arm Event Register

To see where the Arm Event Register appears in the status register hierarchy, see Chapter 41, “Status Reporting,” starting on page 1253.

To unserstand the SCPI status register model, see "Introduction to :STATus Commands" on page 883.

The following bits appear in the Arm Event Register.

Table 115 Arm Event Register Bits

Bit	Name	Description	When Set (1 = High = True), Indicates:
15	---	---	(Not used, always zero.)
14	---	---	(Not used.)
13	---	---	(Not used.)
12	---	---	(Not used.)
11	---	---	(Not used.)
10	---	---	(Not used.)
9	---	---	(Not used.)
8	---	---	(Not used.)
7	---	---	(Not used.)
6	---	---	(Not used.)
5	---	---	(Not used.)
4	---	---	(Not used.)
3	---	---	(Not used.)
2	---	---	(Not used.)
1	---	---	(Not used.)
0	Armed	Armed	The trigger system is ready to look of triggers.

When enabled, the summary message for the Arm Event Register goes to Operation Status Register bit 5 (WAIT TRIG). See ":STATus:OPERation Commands" on page 888.

:STATus:OPERation:ARM:BIT<b>:CONDition?

(see page 1292)

Query Syntax

```
:STATus:OPERation:ARM:BIT<b>:CONDition?

<b> ::= 0-14; an integer in NR1 format
```

The :STATus:OPERation:ARM:BIT<b>:CONDition? query returns the value of a particular bit in the Condition register.

The Condition register continuously monitors status. Reading a Condition register bit does not affect its contents.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:OPERation:ARM:BIT<b>:ENABle" on page 907
- ":STATus:OPERation:ARM:BIT<b>:NTRansition" on page 908
- ":STATus:OPERation:ARM:BIT<b>:PTRansition" on page 909
- ":STATus:OPERation:ARM:BIT<b>[:EVENt]?" on page 910
- ":STATus:OPERation:ARM:CONDition?" on page 911
- ":STATus:OPERation:ARM:ENABle" on page 912
- ":STATus:OPERation:ARM:NTRansition" on page 913
- ":STATus:OPERation:ARM:PTRansition" on page 914
- ":STATus:OPERation:ARM[:EVENt]?" on page 915
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:ARM:BIT<b>:ENABle

 (see page 1292)

Command Syntax

```
:STATus:OPERation:ARM:BIT<b>:ENABle <bit_value>

<b> ::= 0-14; an integer in NR1 format

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

The :STATus:OPERation:ARM:BIT<b>:ENABle command sets a particular bit in the Enable register that identifies a corresponding Event register bit that will be ORed with other identified Event register bits to create the Summary Message bit that is sent to the parent status register.

Query Syntax

```
:STATus:OPERation:ARM:BIT<b>:ENABle?
```

The :STATus:OPERation:ARM:BIT<b>:ENABle? query returns the value of the specified Enable register bit.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:OPERation:ARM:BIT<b>:CONDition?" on page 906
- ":STATus:OPERation:ARM:BIT<b>:NTRansition" on page 908
- ":STATus:OPERation:ARM:BIT<b>:PTRansition" on page 909
- ":STATus:OPERation:ARM:BIT<b>[:EVENt]?" on page 910
- ":STATus:OPERation:ARM:CONDition?" on page 911
- ":STATus:OPERation:ARM:ENABle" on page 912
- ":STATus:OPERation:ARM:NTRansition" on page 913
- ":STATus:OPERation:ARM:PTRansition" on page 914
- ":STATus:OPERation:ARM[:EVENt]?" on page 915
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:ARM:BIT<b>:NTRansition

(see page 1292)

Command Syntax

```
:STATus:OPERation:ARM:BIT<b>:NTRansition <bit_value>

<b> ::= 0-14; an integer in NR1 format

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

The :STATus:OPERation:ARM:BIT<b>:NTRansition command sets a particular bit in the Negative Transition filter that identifies a Condition register bit on which a transition from 1 to 0 or TRUE to FALSE will be latched into the corresponding Event register bit.

Query Syntax

```
:STATus:OPERation:ARM:BIT<b>:NTRansition?
```

The :STATus:OPERation:ARM:BIT<b>:NTRansition? query returns the value of the specified Negative Transition filter bit.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:OPERation:ARM:BIT<b>:CONDition?" on page 906
- ":STATus:OPERation:ARM:BIT<b>:ENABle" on page 907
- ":STATus:OPERation:ARM:BIT<b>:PTRansition" on page 909
- ":STATus:OPERation:ARM:BIT<b>[:EVENt]?" on page 910
- ":STATus:OPERation:ARM:CONDition?" on page 911
- ":STATus:OPERation:ARM:ENABle" on page 912
- ":STATus:OPERation:ARM:NTRansition" on page 913
- ":STATus:OPERation:ARM:PTRansition" on page 914
- ":STATus:OPERation:ARM[:EVENt]?" on page 915
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:ARM:BIT<b>:PTRansition

 (see page 1292)

Command Syntax

```
:STATus:OPERation:ARM:BIT<b>:PTRansition <bit_value>

<b> ::= 0-14; an integer in NR1 format

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

The :STATus:OPERation:ARM:BIT<b>:PTRansition command sets a particular bit in the Positive Transition filter that identifies a Condition register bit on which a transition from 0 to 1 or FALSE to TRUE will be latched into the corresponding Event register bit.

Query Syntax

```
:STATus:OPERation:ARM:BIT<b>:PTRansition?
```

The :STATus:OPERation:ARM:BIT<b>:PTRansition? query returns the value of the specified Positive Transition filter bit.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:OPERation:ARM:BIT<b>:CONDition?" on page 906
- ":STATus:OPERation:ARM:BIT<b>:ENABle" on page 907
- ":STATus:OPERation:ARM:BIT<b>:NTRansition" on page 908
- ":STATus:OPERation:ARM:BIT<b>[:EVENt]?" on page 910
- ":STATus:OPERation:ARM:CONDition?" on page 911
- ":STATus:OPERation:ARM:ENABle" on page 912
- ":STATus:OPERation:ARM:NTRansition" on page 913
- ":STATus:OPERation:ARM:PTRansition" on page 914
- ":STATus:OPERation:ARM[:EVENt]?" on page 915
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:ARM:BIT<b>[:EVENt]?

N (see page 1292)

Query Syntax

```
:STATus:OPERation:ARM:BIT<b>[:EVENt]?

<b> ::= 0-14; an integer in NR1 format
```

The :STATus:OPERation:ARM:BIT<b>[:EVENt]? query returns the value of the specified Event register bit and clears the bit.

NOTE The :AER? query is an alias for the :STATus:OPERation:ARM:BIT0[:EVENt]? query.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:OPERation:ARM:BIT<b>:CONDition?" on page 906
- ":STATus:OPERation:ARM:BIT<b>:ENABle" on page 907
- ":STATus:OPERation:ARM:BIT<b>:NTRansition" on page 908
- ":STATus:OPERation:ARM:BIT<b>:PTRansition" on page 909
- ":STATus:OPERation:ARM:CONDition?" on page 911
- ":STATus:OPERation:ARM:ENABle" on page 912
- ":STATus:OPERation:ARM:NTRansition" on page 913
- ":STATus:OPERation:ARM:PTRansition" on page 914
- ":STATus:OPERation:ARM[:EVENt]?" on page 915
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184
- ":AER? (Arm Event Register)" on page 211

:STATus:OPERation:ARM:CONDition?

N (see page 1292)

Query Syntax `:STATus:OPERation:ARM:CONDition?`

The :STATus:OPERation:ARM:CONDition? query returns the decimal value of the sum of the bits in the Condition register.

The Condition register continuously monitors status. Reading the Condition register does not affect its contents.

Return Format `<sum_of_bits><NL>`

`<sum_of_bits> ::= 0-32767; an integer in NR1 format`

See Also

- ":STATus:OPERation:ARM:BIT<b>:CONDition?" on page 906
- ":STATus:OPERation:ARM:BIT<b>:ENABle" on page 907
- ":STATus:OPERation:ARM:BIT<b>:NTRansition" on page 908
- ":STATus:OPERation:ARM:BIT<b>:PTRansition" on page 909
- ":STATus:OPERation:ARM:BIT<b>[:EVENt]?" on page 910
- ":STATus:OPERation:ARM:ENABle" on page 912
- ":STATus:OPERation:ARM:NTRansition" on page 913
- ":STATus:OPERation:ARM:PTRansition" on page 914
- ":STATus:OPERation:ARM[:EVENt]?" on page 915
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:ARM:ENABle

N (see page 1292)

Command Syntax

```
:STATus:OPERation:ARM:ENABle <sum_of_bits>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

The :STATus:OPERation:ARM:ENABle command sets bits in the Enable register that identify which Event register bits are ORed to create the Summary Message bit that is sent to the parent status register.

Query Syntax

```
:STATus:OPERation:ARM:ENABle?
```

The :STATus:OPERation:ARM:ENABle? query returns the decimal value of the sum of the bits in the Enable register.

Return Format

```
<sum_of_bits><NL>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

See Also

- ":STATus:OPERation:ARM:BIT<b>:CONDition?" on page 906
- ":STATus:OPERation:ARM:BIT<b>:ENABle" on page 907
- ":STATus:OPERation:ARM:BIT<b>:NTRansition" on page 908
- ":STATus:OPERation:ARM:BIT<b>:PTRansition" on page 909
- ":STATus:OPERation:ARM:BIT<b>[:EVENt]?" on page 910
- ":STATus:OPERation:ARM:CONDition?" on page 911
- ":STATus:OPERation:ARM:NTRansition" on page 913
- ":STATus:OPERation:ARM:PTRansition" on page 914
- ":STATus:OPERation:ARM[:EVENt]?" on page 915
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:ARM:NTRansition

 (see page 1292)

Command Syntax

```
:STATus:OPERation:ARM:NTRansition <sum_of_bits>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

The :STATus:OPERation:ARM:NTRansition command sets bits in the Negative Transition filter that identify the Condition register bits on which transitions from 1 to 0 or TRUE to FALSE will be latched into the Event register.

Query Syntax

```
:STATus:OPERation:ARM:NTRansition?
```

The :STATus:OPERation:ARM:NTRansition? query returns the decimal value of the sum of the bits in the Negative Transition filter.

Return Format

```
<sum_of_bits><NL>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

See Also

- ":STATus:OPERation:ARM:BIT<b>:CONDition?" on page 906
- ":STATus:OPERation:ARM:BIT<b>:ENABle" on page 907
- ":STATus:OPERation:ARM:BIT<b>:NTRansition" on page 908
- ":STATus:OPERation:ARM:BIT<b>:PTRansition" on page 909
- ":STATus:OPERation:ARM:BIT<b>[:EVENt]?" on page 910
- ":STATus:OPERation:ARM:CONDition?" on page 911
- ":STATus:OPERation:ARM:ENABle" on page 912
- ":STATus:OPERation:ARM:PTRansition" on page 914
- ":STATus:OPERation:ARM[:EVENt]?" on page 915
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:ARM:PTRansition

 (see page 1292)

Command Syntax

```
:STATus:OPERation:ARM:PTRansition <sum_of_bits>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

The :STATus:OPERation:ARM:PTRansition command sets bits in the Positive Transition filter that identify the Condition register bits on which transitions from 0 to 1 or FALSE to TRUE will be latched into the Event register.

Query Syntax

```
:STATus:OPERation:ARM:PTRansition?
```

The :STATus:OPERation:ARM:PTRansition? query returns the decimal value of the sum of the bits in the Positive Transition filter.

Return Format

```
<sum_of_bits><NL>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

See Also

- ":STATus:OPERation:ARM:BIT<b>:CONDition?" on page 906
- ":STATus:OPERation:ARM:BIT<b>:ENABle" on page 907
- ":STATus:OPERation:ARM:BIT<b>:NTRansition" on page 908
- ":STATus:OPERation:ARM:BIT<b>:PTRansition" on page 909
- ":STATus:OPERation:ARM:BIT<b>[:EVENt]?" on page 910
- ":STATus:OPERation:ARM:CONDition?" on page 911
- ":STATus:OPERation:ARM:ENABle" on page 912
- ":STATus:OPERation:ARM:NTRansition" on page 913
- ":STATus:OPERation:ARM[:EVENt]?" on page 915
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:ARM[:EVENt]?

(see page 1292)

Query Syntax `:STATus:OPERation:ARM[:EVENt]?`

The :STATus:OPERation:ARM[:EVENt]? query returns the decimal value of the sum of the bits in the Event register and clears the register.

The contents of the Event register show the events that have occurred since the last time the register was read.

NOTE **Because bit 0 is the only bit being used in the Arm Event Register and the only one that can be latched into the Event register, the :AER? query is an alias for the :STATus:OPERation:ARM[:EVENt]? query.**

Return Format `<sum_of_bits><NL>`

`<sum_of_bits> ::= 0-32767; an integer in NR1 format`

See Also
- ":STATus:OPERation:ARM:BIT<b>:CONDition?" on page 906
- ":STATus:OPERation:ARM:BIT<b>:ENABle" on page 907
- ":STATus:OPERation:ARM:BIT<b>:NTRansition" on page 908
- ":STATus:OPERation:ARM:BIT<b>:PTRansition" on page 909
- ":STATus:OPERation:ARM:BIT<b>[:EVENt]?" on page 910
- ":STATus:OPERation:ARM:CONDition?" on page 911
- ":STATus:OPERation:ARM:ENABle" on page 912
- ":STATus:OPERation:ARM:NTRansition" on page 913
- ":STATus:OPERation:ARM:PTRansition" on page 914
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184
- ":AER? (Arm Event Register)" on page 211

:STATus:OPERation:HARDware Commands

The :STATus:OPERation:HARDware commands control the Hardware Operation Register.

Table 116 :STATus:OPERation:HARDware Commands Summary

Command	Query	Options and Query Returns
n/a	:STATus:OPERation:HARDware:BIT<b>:CONDition? (see page 919)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:HARDware:BIT<b>:ENABle <bit_value> (see page 920)	:STATus:OPERation:HARDware:BIT<b>:ENABle? (see page 920)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:HARDware:BIT<b>:NTRansition <bit_value> (see page 921)	:STATus:OPERation:HARDware:BIT<b>:NTRansition? (see page 921)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:HARDware:BIT<b>:PTRansition <bit_value> (see page 922)	:STATus:OPERation:HARDware:BIT<b>:PTRansition? (see page 922)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation:HARDware:BIT<b>[:EVENt]? (see page 923)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation:HARDware:CONDition? (see page 924)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:HARDware:ENABle <sum_of_bits> (see page 925)	:STATus:OPERation:HARDware:ENABle? (see page 925)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:HARDware:NTRansition <sum_of_bits> (see page 926)	:STATus:OPERation:HARDware:NTRansition? (see page 926)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format

Table 116 :STATus:OPERation:HARDware Commands Summary (continued)

Command	Query	Options and Query Returns
:STATus:OPERation:HARDware:PTRansition <sum_of_bits> (see page 927)	:STATus:OPERation:HARDware:PTRansition? (see page 927)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation:HARDware[:EVENt]? (see page 928)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format

Hardware Operation Register

To see where the Hardware Operation Register appears in the status register hierarchy, see Chapter 41, “Status Reporting,” starting on page 1253.

To unserstand the SCPI status register model, see "Introduction to :STATus Commands" on page 883.

The following bits appear in the Hardware Operation Register.

Table 117 Hardware Operation Register Bits

Bit	Name	Description	When Set (1 = High = True), Indicates:
15	---	---	(Not used, always zero.)
14	---	---	(Not used.)
13	---	---	(Not used.)
12	PLL Locked	PLL Locked	This bit is for internal use and is not intended for general use.
11	---	---	(Not used.)
10	---	---	(Not used.)
9	---	---	(Not used.)
8	---	---	(Not used.)
7	---	---	(Not used.)
6	---	---	(Not used.)
5	---	---	(Not used.)
4	---	---	(Not used.)
3	---	---	(Not used.)
2	---	---	(Not used.)
1	Battery Low	Battery Low	Reserved for battery-operated oscilloscope models.
0	Battery On	Battery On	Reserved for battery-operated oscilloscope models.

When enabled, the summary message for the Hardware Operation Register goes to Operation Status Register bit 12 (HWE). See ":STATus:OPERation Commands" on page 888.

:STATus:OPERation:HARDware:BIT<b>:CONDition?

N (see page 1292)

Query Syntax

```
:STATus:OPERation:HARDware:BIT<b>:CONDition?

<b> ::= 0-14; an integer in NR1 format
```

The :STATus:OPERation:HARDware:BIT<b>:CONDition? query returns the value of a particular bit in the Condition register.

The Condition register continuously monitors status. Reading a Condition register bit does not affect its contents.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:OPERation:HARDware:BIT<b>:ENABle" on page 920
- ":STATus:OPERation:HARDware:BIT<b>:NTRansition" on page 921
- ":STATus:OPERation:HARDware:BIT<b>:PTRansition" on page 922
- ":STATus:OPERation:HARDware:BIT<b>[:EVENt]?" on page 923
- ":STATus:OPERation:HARDware:CONDition?" on page 924
- ":STATus:OPERation:HARDware:ENABle" on page 925
- ":STATus:OPERation:HARDware:NTRansition" on page 926
- ":STATus:OPERation:HARDware:PTRansition" on page 927
- ":STATus:OPERation:HARDware[:EVENt]?" on page 928
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:HARDware:BIT<b>:ENABle

 (see page 1292)

Command Syntax

```
:STATus:OPERation:HARDware:BIT<b>:ENABle <bit_value>

<b> ::= 0-14; an integer in NR1 format

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

The :STATus:OPERation:HARDware:BIT<b>:ENABle command sets a particular bit in the Enable register that identifies a corresponding Event register bit that will be ORed with other identified Event register bits to create the Summary Message bit that is sent to the parent status register.

Query Syntax

```
:STATus:OPERation:HARDware:BIT<b>:ENABle?
```

The :STATus:OPERation:HARDware:BIT<b>:ENABle? query returns the value of the specified Enable register bit.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:OPERation:HARDware:BIT<b>:CONDition?" on page 919
- ":STATus:OPERation:HARDware:BIT<b>:NTRansition" on page 921
- ":STATus:OPERation:HARDware:BIT<b>:PTRansition" on page 922
- ":STATus:OPERation:HARDware:BIT<b>[:EVENt]?" on page 923
- ":STATus:OPERation:HARDware:CONDition?" on page 924
- ":STATus:OPERation:HARDware:ENABle" on page 925
- ":STATus:OPERation:HARDware:NTRansition" on page 926
- ":STATus:OPERation:HARDware:PTRansition" on page 927
- ":STATus:OPERation:HARDware[:EVENt]?" on page 928
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:HARDware:BIT<b>:NTRansition

N (see page 1292)

Command Syntax

```
:STATus:OPERation:HARDware:BIT<b>:NTRansition <bit_value>

<b> ::= 0-14; an integer in NR1 format

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

The :STATus:OPERation:HARDware:BIT<b>:NTRansition command sets a particular bit in the Negative Transition filter that identifies a Condition register bit on which a transition from 1 to 0 or TRUE to FALSE will be latched into the corresponding Event register bit.

Query Syntax

```
:STATus:OPERation:HARDware:BIT<b>:NTRansition?
```

The :STATus:OPERation:HARDware:BIT<b>:NTRansition? query returns the value of the specified Negative Transition filter bit.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:OPERation:HARDware:BIT<b>:CONDition?" on page 919
- ":STATus:OPERation:HARDware:BIT<b>:ENABle" on page 920
- ":STATus:OPERation:HARDware:BIT<b>:PTRansition" on page 922
- ":STATus:OPERation:HARDware:BIT<b>[:EVENt]?" on page 923
- ":STATus:OPERation:HARDware:CONDition?" on page 924
- ":STATus:OPERation:HARDware:ENABle" on page 925
- ":STATus:OPERation:HARDware:NTRansition" on page 926
- ":STATus:OPERation:HARDware:PTRansition" on page 927
- ":STATus:OPERation:HARDware[:EVENt]?" on page 928
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:HARDware:BIT<b>:PTRansition

N (see page 1292)

Command Syntax

```
:STATus:OPERation:HARDware:BIT<b>:PTRansition <bit_value>

<b> ::= 0-14; an integer in NR1 format

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

The :STATus:OPERation:HARDware:BIT<b>:PTRansition command sets a particular bit in the Positive Transition filter that identifies a Condition register bit on which a transition from 0 to 1 or FALSE to TRUE will be latched into the corresponding Event register bit.

Query Syntax

```
:STATus:OPERation:HARDware:BIT<b>:PTRansition?
```

The :STATus:OPERation:HARDware:BIT<b>:PTRansition? query returns the value of the specified Positive Transition filter bit.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:OPERation:HARDware:BIT<b>:CONDition?" on page 919
- ":STATus:OPERation:HARDware:BIT<b>:ENABle" on page 920
- ":STATus:OPERation:HARDware:BIT<b>:NTRansition" on page 921
- ":STATus:OPERation:HARDware:BIT<b>[:EVENt]?" on page 923
- ":STATus:OPERation:HARDware:CONDition?" on page 924
- ":STATus:OPERation:HARDware:ENABle" on page 925
- ":STATus:OPERation:HARDware:NTRansition" on page 926
- ":STATus:OPERation:HARDware:PTRansition" on page 927
- ":STATus:OPERation:HARDware[:EVENt]?" on page 928
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:HARDware:BIT<b>[:EVENt]?

N (see page 1292)

Query Syntax

```
:STATus:OPERation:HARDware:BIT<b>[:EVENt]?

<b> ::= 0-14; an integer in NR1 format
```

The :STATus:OPERation:HARDware:BIT<b>[:EVENt]? query returns the value of the specified Event register bit and clears the bit.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:OPERation:HARDware:BIT<b>:CONDition?" on page 919
- ":STATus:OPERation:HARDware:BIT<b>:ENABle" on page 920
- ":STATus:OPERation:HARDware:BIT<b>:NTRansition" on page 921
- ":STATus:OPERation:HARDware:BIT<b>:PTRansition" on page 922
- ":STATus:OPERation:HARDware:CONDition?" on page 924
- ":STATus:OPERation:HARDware:ENABle" on page 925
- ":STATus:OPERation:HARDware:NTRansition" on page 926
- ":STATus:OPERation:HARDware:PTRansition" on page 927
- ":STATus:OPERation:HARDware[:EVENt]?" on page 928
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:HARDware:CONDition?

(see page 1292)

Query Syntax `:STATus:OPERation:HARDware:CONDition?`

The :STATus:OPERation:HARDware:CONDition? query returns the decimal value of the sum of the bits in the Condition register.

The Condition register continuously monitors status. Reading the Condition register does not affect its contents.

Return Format `<sum_of_bits><NL>`

`<sum_of_bits> ::= 0-32767; an integer in NR1 format`

See Also

- ":STATus:OPERation:HARDware:BIT<b>:CONDition?" on page 919
- ":STATus:OPERation:HARDware:BIT<b>:ENABle" on page 920
- ":STATus:OPERation:HARDware:BIT<b>:NTRansition" on page 921
- ":STATus:OPERation:HARDware:BIT<b>:PTRansition" on page 922
- ":STATus:OPERation:HARDware:BIT<b>[:EVENt]?" on page 923
- ":STATus:OPERation:HARDware:ENABle" on page 925
- ":STATus:OPERation:HARDware:NTRansition" on page 926
- ":STATus:OPERation:HARDware:PTRansition" on page 927
- ":STATus:OPERation:HARDware[:EVENt]?" on page 928
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:HARDware:ENABle

(see page 1292)

Command Syntax

```
:STATus:OPERation:HARDware:ENABle <sum_of_bits>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

The :STATus:OPERation:HARDware:ENABle command sets bits in the Enable register that identify which Event register bits are ORed to create the Summary Message bit that is sent to the parent status register.

Query Syntax

```
:STATus:OPERation:HARDware:ENABle?
```

The :STATus:OPERation:HARDware:ENABle? query returns the decimal value of the sum of the bits in the Enable register.

Return Format

```
<sum_of_bits><NL>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

See Also

- ":STATus:OPERation:HARDware:BIT<b>:CONDition?" on page 919
- ":STATus:OPERation:HARDware:BIT<b>:ENABle" on page 920
- ":STATus:OPERation:HARDware:BIT<b>:NTRansition" on page 921
- ":STATus:OPERation:HARDware:BIT<b>:PTRansition" on page 922
- ":STATus:OPERation:HARDware:BIT<b>[:EVENt]?" on page 923
- ":STATus:OPERation:HARDware:CONDition?" on page 924
- ":STATus:OPERation:HARDware:NTRansition" on page 926
- ":STATus:OPERation:HARDware:PTRansition" on page 927
- ":STATus:OPERation:HARDware[:EVENt]?" on page 928
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:HARDware:NTRansition

N (see page 1292)

Command Syntax

```
:STATus:OPERation:HARDware:NTRansition <sum_of_bits>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

The :STATus:OPERation:HARDware:NTRansition command sets bits in the Negative Transition filter that identify the Condition register bits on which transitions from 1 to 0 or TRUE to FALSE will be latched into the Event register.

Query Syntax

```
:STATus:OPERation:HARDware:NTRansition?
```

The :STATus:OPERation:HARDware:NTRansition? query returns the decimal value of the sum of the bits in the Negative Transition filter.

Return Format

```
<sum_of_bits><NL>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

See Also

- ":STATus:OPERation:HARDware:BIT<b>:CONDition?" on page 919
- ":STATus:OPERation:HARDware:BIT<b>:ENABle" on page 920
- ":STATus:OPERation:HARDware:BIT<b>:NTRansition" on page 921
- ":STATus:OPERation:HARDware:BIT<b>:PTRansition" on page 922
- ":STATus:OPERation:HARDware:BIT<b>[:EVENt]?" on page 923
- ":STATus:OPERation:HARDware:CONDition?" on page 924
- ":STATus:OPERation:HARDware:ENABle" on page 925
- ":STATus:OPERation:HARDware:PTRansition" on page 927
- ":STATus:OPERation:HARDware[:EVENt]?" on page 928
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:HARDware:PTRansition

 (see page 1292)

Command Syntax

```
:STATus:OPERation:HARDware:PTRansition <sum_of_bits>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

The :STATus:OPERation:HARDware:PTRansition command sets bits in the Positive Transition filter that identify the Condition register bits on which transitions from 0 to 1 or FALSE to TRUE will be latched into the Event register.

Query Syntax

```
:STATus:OPERation:HARDware:PTRansition?
```

The :STATus:OPERation:HARDware:PTRansition? query returns the decimal value of the sum of the bits in the Positive Transition filter.

Return Format

```
<sum_of_bits><NL>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

See Also

- ":STATus:OPERation:HARDware:BIT<b>:CONDition?" on page 919
- ":STATus:OPERation:HARDware:BIT<b>:ENABle" on page 920
- ":STATus:OPERation:HARDware:BIT<b>:NTRansition" on page 921
- ":STATus:OPERation:HARDware:BIT<b>:PTRansition" on page 922
- ":STATus:OPERation:HARDware:BIT<b>[:EVENt]?" on page 923
- ":STATus:OPERation:HARDware:CONDition?" on page 924
- ":STATus:OPERation:HARDware:ENABle" on page 925
- ":STATus:OPERation:HARDware:NTRansition" on page 926
- ":STATus:OPERation:HARDware[:EVENt]?" on page 928
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:HARDware[:EVENt]?

(see page 1292)

Query Syntax `:STATus:OPERation:HARDware[:EVENt]?`

The :STATus:OPERation:HARDware[:EVENt]? query returns the decimal value of the sum of the bits in the Event register and clears the register.

The contents of the Event register show the events that have occurred since the last time the register was read.

Return Format `<sum_of_bits><NL>`

`<sum_of_bits> ::= 0-32767; an integer in NR1 format`

See Also

- ":STATus:OPERation:HARDware:BIT<b>:CONDition?" on page 919
- ":STATus:OPERation:HARDware:BIT<b>:ENABle" on page 920
- ":STATus:OPERation:HARDware:BIT<b>:NTRansition" on page 921
- ":STATus:OPERation:HARDware:BIT<b>:PTRansition" on page 922
- ":STATus:OPERation:HARDware:BIT<b>[:EVENt]?" on page 923
- ":STATus:OPERation:HARDware:CONDition?" on page 924
- ":STATus:OPERation:HARDware:ENABle" on page 925
- ":STATus:OPERation:HARDware:NTRansition" on page 926
- ":STATus:OPERation:HARDware:PTRansition" on page 927
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:LOCal Commands

The :STATus:OPERation:LOCal commands control the Local Operation Register.

Table 118 :STATus:OPERation:LOCal Commands Summary

Command	Query	Options and Query Returns
n/a	:STATus:OPERation:LOCal:BIT<b>:CONDition? (see page 932)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:LOCal:BIT<b>:ENABle <bit_value> (see page 933)	:STATus:OPERation:LOCal:BIT<b>:ENABle? (see page 933)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:LOCal:BIT<b>:NTRansition <bit_value> (see page 934)	:STATus:OPERation:LOCal:BIT<b>:NTRansition? (see page 934)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:LOCal:BIT<b>:PTRansition <bit_value> (see page 935)	:STATus:OPERation:LOCal:BIT<b>:PTRansition? (see page 935)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation:LOCal:BIT<b>[:EVENt]? (see page 936)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation:LOCal:CONDition? (see page 937)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:LOCal:ENABle <sum_of_bits> (see page 938)	:STATus:OPERation:LOCal:ENABle? (see page 938)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:LOCal:NTRansition <sum_of_bits> (see page 939)	:STATus:OPERation:LOCal:NTRansition? (see page 939)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format

Table 118 :STATus:OPERation:LOCal Commands Summary (continued)

Command	Query	Options and Query Returns
:STATus:OPERation:LOCal:PTRansition <sum_of_bits> (see page 940)	:STATus:OPERation:LOCal:PTRansition? (see page 940)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation:LOCal[:EVENt]? (see page 941)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format

Local Operation Register

To see where the Local Operation Register appears in the status register hierarchy, see Chapter 41, “Status Reporting,” starting on page 1253.

To unserstand the SCPI status register model, see "Introduction to :STATus Commands" on page 883.

The following bits appear in the Local Operation Register.

Table 119 Local Operation Register Bits

Bit	Name	Description	When Set (1 = High = True), Indicates:
15	---	---	(Not used, always zero.)
14	---	---	(Not used.)
13	---	---	(Not used.)
12	---	---	(Not used.)
11	---	---	(Not used.)
10	---	---	(Not used.)
9	---	---	(Not used.)
8	---	---	(Not used.)
7	---	---	(Not used.)
6	---	---	(Not used.)
5	---	---	(Not used.)
4	---	---	(Not used.)
3	---	---	(Not used.)
2	---	---	(Not used.)
1	Test Fail	Test Fail	A test failure has occurred.
0	Test Complete	Test Complete	A test completion has occurred.

When enabled, the summary message for the Local Operation Register goes to Operation Status Register bit 8 (Lcl). See ":STATus:OPERation Commands" on page 888.

:STATus:OPERation:LOCal:BIT<b>:CONDition?

N (see page 1292)

Query Syntax

```
:STATus:OPERation:LOCal:BIT<b>:CONDition?

<b> ::= 0-14; an integer in NR1 format
```

The :STATus:OPERation:LOCal:BIT<b>:CONDition? query returns the value of a particular bit in the Condition register.

The Condition register continuously monitors status. Reading a Condition register bit does not affect its contents.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:OPERation:LOCal:BIT<b>:ENABle" on page 933
- ":STATus:OPERation:LOCal:BIT<b>:NTRansition" on page 934
- ":STATus:OPERation:LOCal:BIT<b>:PTRansition" on page 935
- ":STATus:OPERation:LOCal:BIT<b>[:EVENt]?" on page 936
- ":STATus:OPERation:LOCal:CONDition?" on page 937
- ":STATus:OPERation:LOCal:ENABle" on page 938
- ":STATus:OPERation:LOCal:NTRansition" on page 939
- ":STATus:OPERation:LOCal:PTRansition" on page 940
- ":STATus:OPERation:LOCal[:EVENt]?" on page 941
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:LOCal:BIT<b>:ENABle

N (see page 1292)

Command Syntax

```
:STATus:OPERation:LOCal:BIT<b>:ENABle <bit_value>

<b> ::= 0-14; an integer in NR1 format

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

The :STATus:OPERation:LOCal:BIT<b>:ENABle command sets a particular bit in the Enable register that identifies a corresponding Event register bit that will be ORed with other identified Event register bits to create the Summary Message bit that is sent to the parent status register.

Query Syntax

```
:STATus:OPERation:LOCal:BIT<b>:ENABle?
```

The :STATus:OPERation:LOCal:BIT<b>:ENABle? query returns the value of the specified Enable register bit.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:OPERation:LOCal:BIT<b>:CONDition?" on page 932
- ":STATus:OPERation:LOCal:BIT<b>:NTRansition" on page 934
- ":STATus:OPERation:LOCal:BIT<b>:PTRansition" on page 935
- ":STATus:OPERation:LOCal:BIT<b>[:EVENt]?" on page 936
- ":STATus:OPERation:LOCal:CONDition?" on page 937
- ":STATus:OPERation:LOCal:ENABle" on page 938
- ":STATus:OPERation:LOCal:NTRansition" on page 939
- ":STATus:OPERation:LOCal:PTRansition" on page 940
- ":STATus:OPERation:LOCal[:EVENt]?" on page 941
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:LOCal:BIT<b>:NTRansition

(see page 1292)

Command Syntax

```
:STATus:OPERation:LOCal:BIT<b>:NTRansition <bit_value>

<b> ::= 0-14; an integer in NR1 format

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

The :STATus:OPERation:LOCal:BIT<b>:NTRansition command sets a particular bit in the Negative Transition filter that identifies a Condition register bit on which a transition from 1 to 0 or TRUE to FALSE will be latched into the corresponding Event register bit.

Query Syntax

```
:STATus:OPERation:LOCal:BIT<b>:NTRansition?
```

The :STATus:OPERation:LOCal:BIT<b>:NTRansition? query returns the value of the specified Negative Transition filter bit.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:OPERation:LOCal:BIT<b>:CONDition?" on page 932
- ":STATus:OPERation:LOCal:BIT<b>:ENABle" on page 933
- ":STATus:OPERation:LOCal:BIT<b>:PTRansition" on page 935
- ":STATus:OPERation:LOCal:BIT<b>[:EVENt]?" on page 936
- ":STATus:OPERation:LOCal:CONDition?" on page 937
- ":STATus:OPERation:LOCal:ENABle" on page 938
- ":STATus:OPERation:LOCal:NTRansition" on page 939
- ":STATus:OPERation:LOCal:PTRansition" on page 940
- ":STATus:OPERation:LOCal[:EVENt]?" on page 941
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:LOCal:BIT<b>:PTRansition

(see page 1292)

Command Syntax

```
:STATus:OPERation:LOCal:BIT<b>:PTRansition <bit_value>

<b> ::= 0-14; an integer in NR1 format

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

The :STATus:OPERation:LOCal:BIT<b>:PTRansition command sets a particular bit in the Positive Transition filter that identifies a Condition register bit on which a transition from 0 to 1 or FALSE to TRUE will be latched into the corresponding Event register bit.

Query Syntax

```
:STATus:OPERation:LOCal:BIT<b>:PTRansition?
```

The :STATus:OPERation:LOCal:BIT<b>:PTRansition? query returns the value of the specified Positive Transition filter bit.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:OPERation:LOCal:BIT<b>:CONDition?" on page 932
- ":STATus:OPERation:LOCal:BIT<b>:ENABle" on page 933
- ":STATus:OPERation:LOCal:BIT<b>:NTRansition" on page 934
- ":STATus:OPERation:LOCal:BIT<b>[:EVENt]?" on page 936
- ":STATus:OPERation:LOCal:CONDition?" on page 937
- ":STATus:OPERation:LOCal:ENABle" on page 938
- ":STATus:OPERation:LOCal:NTRansition" on page 939
- ":STATus:OPERation:LOCal:PTRansition" on page 940
- ":STATus:OPERation:LOCal[:EVENt]?" on page 941
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:LOCal:BIT<b>[:EVENt]?

N (see page 1292)

Query Syntax

```
:STATus:OPERation:LOCal:BIT<b>[:EVENt]?

<b> ::= 0-14; an integer in NR1 format
```

The :STATus:OPERation:LOCal:BIT<b>[:EVENt]? query returns the value of the specified Event register bit and clears the bit.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:OPERation:LOCal:BIT<b>:CONDition?" on page 932
- ":STATus:OPERation:LOCal:BIT<b>:ENABle" on page 933
- ":STATus:OPERation:LOCal:BIT<b>:NTRansition" on page 934
- ":STATus:OPERation:LOCal:BIT<b>:PTRansition" on page 935
- ":STATus:OPERation:LOCal:CONDition?" on page 937
- ":STATus:OPERation:LOCal:ENABle" on page 938
- ":STATus:OPERation:LOCal:NTRansition" on page 939
- ":STATus:OPERation:LOCal:PTRansition" on page 940
- ":STATus:OPERation:LOCal[:EVENt]?" on page 941
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:LOCal:CONDition?

(see page 1292)

Query Syntax `:STATus:OPERation:LOCal:CONDition?`

The :STATus:OPERation:LOCal:CONDition? query returns the decimal value of the sum of the bits in the Condition register.

The Condition register continuously monitors status. Reading the Condition register does not affect its contents.

Return Format `<sum_of_bits><NL>`

`<sum_of_bits> ::= 0-32767; an integer in NR1 format`

See Also

- ":STATus:OPERation:LOCal:BIT<b>:CONDition?" on page 932
- ":STATus:OPERation:LOCal:BIT<b>:ENABle" on page 933
- ":STATus:OPERation:LOCal:BIT<b>:NTRansition" on page 934
- ":STATus:OPERation:LOCal:BIT<b>:PTRansition" on page 935
- ":STATus:OPERation:LOCal:BIT<b>[:EVENt]?" on page 936
- ":STATus:OPERation:LOCal:ENABle" on page 938
- ":STATus:OPERation:LOCal:NTRansition" on page 939
- ":STATus:OPERation:LOCal:PTRansition" on page 940
- ":STATus:OPERation:LOCal[:EVENt]?" on page 941
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:LOCal:ENABle

(see page 1292)

Command Syntax

```
:STATus:OPERation:LOCal:ENABle <sum_of_bits>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

The :STATus:OPERation:LOCal:ENABle command sets bits in the Enable register that identify which Event register bits are ORed to create the Summary Message bit that is sent to the parent status register.

Query Syntax

```
:STATus:OPERation:LOCal:ENABle?
```

The :STATus:OPERation:LOCal:ENABle? query returns the decimal value of the sum of the bits in the Enable register.

Return Format

```
<sum_of_bits><NL>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

See Also

- ":STATus:OPERation:LOCal:BIT<b>:CONDition?" on page 932
- ":STATus:OPERation:LOCal:BIT<b>:ENABle" on page 933
- ":STATus:OPERation:LOCal:BIT<b>:NTRansition" on page 934
- ":STATus:OPERation:LOCal:BIT<b>:PTRansition" on page 935
- ":STATus:OPERation:LOCal:BIT<b>[:EVENt]?" on page 936
- ":STATus:OPERation:LOCal:CONDition?" on page 937
- ":STATus:OPERation:LOCal:NTRansition" on page 939
- ":STATus:OPERation:LOCal:PTRansition" on page 940
- ":STATus:OPERation:LOCal[:EVENt]?" on page 941
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:LOCal:NTRansition

 (see page 1292)

Command Syntax

```
:STATus:OPERation:LOCal:NTRansition <sum_of_bits>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

The :STATus:OPERation:LOCal:NTRansition command sets bits in the Negative Transition filter that identify the Condition register bits on which transitions from 1 to 0 or TRUE to FALSE will be latched into the Event register.

Query Syntax

```
:STATus:OPERation:LOCal:NTRansition?
```

The :STATus:OPERation:LOCal:NTRansition? query returns the decimal value of the sum of the bits in the Negative Transition filter.

Return Format

```
<sum_of_bits><NL>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

See Also

- ":STATus:OPERation:LOCal:BIT<b>:CONDition?" on page 932
- ":STATus:OPERation:LOCal:BIT<b>:ENABle" on page 933
- ":STATus:OPERation:LOCal:BIT<b>:NTRansition" on page 934
- ":STATus:OPERation:LOCal:BIT<b>:PTRansition" on page 935
- ":STATus:OPERation:LOCal:BIT<b>[:EVENt]?" on page 936
- ":STATus:OPERation:LOCal:CONDition?" on page 937
- ":STATus:OPERation:LOCal:ENABle" on page 938
- ":STATus:OPERation:LOCal:PTRansition" on page 940
- ":STATus:OPERation:LOCal[:EVENt]?" on page 941
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:LOCal:PTRansition

(see page 1292)

Command Syntax

```
:STATus:OPERation:LOCal:PTRansition <sum_of_bits>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

The :STATus:OPERation:LOCal:PTRansition command sets bits in the Positive Transition filter that identify the Condition register bits on which transitions from 0 to 1 or FALSE to TRUE will be latched into the Event register.

Query Syntax

```
:STATus:OPERation:LOCal:PTRansition?
```

The :STATus:OPERation:LOCal:PTRansition? query returns the decimal value of the sum of the bits in the Positive Transition filter.

Return Format

```
<sum_of_bits><NL>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

See Also

- ":STATus:OPERation:LOCal:BIT<b>:CONDition?" on page 932
- ":STATus:OPERation:LOCal:BIT<b>:ENABle" on page 933
- ":STATus:OPERation:LOCal:BIT<b>:NTRansition" on page 934
- ":STATus:OPERation:LOCal:BIT<b>:PTRansition" on page 935
- ":STATus:OPERation:LOCal:BIT<b>[:EVENt]?" on page 936
- ":STATus:OPERation:LOCal:CONDition?" on page 937
- ":STATus:OPERation:LOCal:ENABle" on page 938
- ":STATus:OPERation:LOCal:NTRansition" on page 939
- ":STATus:OPERation:LOCal[:EVENt]?" on page 941
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:LOCal[:EVENt]?

 (see page 1292)

Query Syntax `:STATus:OPERation:LOCal[:EVENt]?`

The :STATus:OPERation:LOCal[:EVENt]? query returns the decimal value of the sum of the bits in the Event register and clears the register.

The contents of the Event register show the events that have occurred since the last time the register was read.

Return Format `<sum_of_bits><NL>`

`<sum_of_bits> ::= 0-32767; an integer in NR1 format`

See Also

- ":STATus:OPERation:LOCal:BIT<b>:CONDition?" on page 932
- ":STATus:OPERation:LOCal:BIT<b>:ENABle" on page 933
- ":STATus:OPERation:LOCal:BIT<b>:NTRansition" on page 934
- ":STATus:OPERation:LOCal:BIT<b>:PTRansition" on page 935
- ":STATus:OPERation:LOCal:BIT<b>[:EVENt]?" on page 936
- ":STATus:OPERation:LOCal:CONDition?" on page 937
- ":STATus:OPERation:LOCal:ENABle" on page 938
- ":STATus:OPERation:LOCal:NTRansition" on page 939
- ":STATus:OPERation:LOCal:PTRansition" on page 940
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:MTESt Commands

The :STATus:OPERation:MTESt commands control the Mask Test Operation Register.

Table 120 :STATus:OPERation:MTESt Commands Summary

Command	Query	Options and Query Returns
n/a	:STATus:OPERation:MTESt:BIT<b>:CONDition? (see page 945)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:MTESt:BIT<b>:ENABle <bit_value> (see page 946)	:STATus:OPERation:MTESt:BIT<b>:ENABle? (see page 946)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:MTESt:BIT<b>:NTRansition <bit_value> (see page 947)	:STATus:OPERation:MTESt:BIT<b>:NTRansition? (see page 947)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:MTESt:BIT<b>:PTRansition <bit_value> (see page 948)	:STATus:OPERation:MTESt:BIT<b>:PTRansition? (see page 948)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation:MTESt:BIT<b>[:EVENt]? (see page 949)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation:MTESt:CONDition? (see page 950)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:MTESt:ENABle <sum_of_bits> (see page 951)	:STATus:OPERation:MTESt:ENABle? (see page 951)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:MTESt:NTRansition <sum_of_bits> (see page 952)	:STATus:OPERation:MTESt:NTRansition? (see page 952)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format

Table 120 :STATus:OPERation:MTESt Commands Summary (continued)

Command	Query	Options and Query Returns
:STATus:OPERation:MTESt:PTRansition <sum_of_bits> (see page 953)	:STATus:OPERation:MTESt:PTRansition? (see page 953)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation:MTESt[:EVENt]? (see page 954)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format

Mask Test Operation Register

To see where the Mask Test Operation Register appears in the status register hierarchy, see Chapter 41, “Status Reporting,” starting on page 1253.

To unserstand the SCPI status register model, see "Introduction to :STATus Commands" on page 883.

The following bits appear in the Mask Test Operation Register.

Table 121 Mask Test Operation Register Bits

Bit	Name	Description	When Set (1 = High = True), Indicates:
15	---	---	(Not used, always zero.)
14	---	---	(Not used.)
13	---	---	(Not used.)
12	---	---	(Not used.)
11	---	---	(Not used.)
10	Auto Mask	Auto Mask Created	Auto mask creation completed.
9	Pass	Mask Test Pass	The mask test passed.
8	Started	Mask Testing Started	Mask testing started.
7	---	---	(Not used.)
6	---	---	(Not used.)
5	---	---	(Not used.)
4	---	---	(Not used.)
3	---	---	(Not used.)
2	---	---	(Not used.)
1	Fail	Mask Test Fail	The mask test failed.
0	Complete	Mask Test Complete	The mask test is complete.

When enabled, the summary message for the Mask Test Operation Register goes to Operation Status Register bit 9 (MTE). See ":STATus:OPERation Commands" on page 888.

:STATus:OPERation:MTESt:BIT<b>:CONDition?

N (see page 1292)

Query Syntax

```
:STATus:OPERation:MTESt:BIT<b>:CONDition?

<b> ::= 0-14; an integer in NR1 format
```

The :STATus:OPERation:MTESt:BIT<b>:CONDition? query returns the value of a particular bit in the Condition register.

The Condition register continuously monitors status. Reading a Condition register bit does not affect its contents.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:OPERation:MTESt:BIT<b>:ENABle" on page 946
- ":STATus:OPERation:MTESt:BIT<b>:NTRansition" on page 947
- ":STATus:OPERation:MTESt:BIT<b>:PTRansition" on page 948
- ":STATus:OPERation:MTESt:BIT<b>[:EVENt]?" on page 949
- ":STATus:OPERation:MTESt:CONDition?" on page 950
- ":STATus:OPERation:MTESt:ENABle" on page 951
- ":STATus:OPERation:MTESt:NTRansition" on page 952
- ":STATus:OPERation:MTESt:PTRansition" on page 953
- ":STATus:OPERation:MTESt[:EVENt]?" on page 954
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:MTESt:BIT<b>:ENABle

(see page 1292)

Command Syntax

```
:STATus:OPERation:MTESt:BIT<b>:ENABle <bit_value>

<b> ::= 0-14; an integer in NR1 format

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

The :STATus:OPERation:MTESt:BIT<b>:ENABle command sets a particular bit in the Enable register that identifies a corresponding Event register bit that will be ORed with other identified Event register bits to create the Summary Message bit that is sent to the parent status register.

Query Syntax

```
:STATus:OPERation:MTESt:BIT<b>:ENABle?
```

The :STATus:OPERation:MTESt:BIT<b>:ENABle? query returns the value of the specified Enable register bit.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:OPERation:MTESt:BIT<b>:CONDition?" on page 945
- ":STATus:OPERation:MTESt:BIT<b>:NTRansition" on page 947
- ":STATus:OPERation:MTESt:BIT<b>:PTRansition" on page 948
- ":STATus:OPERation:MTESt:BIT<b>[:EVENt]?" on page 949
- ":STATus:OPERation:MTESt:CONDition?" on page 950
- ":STATus:OPERation:MTESt:ENABle" on page 951
- ":STATus:OPERation:MTESt:NTRansition" on page 952
- ":STATus:OPERation:MTESt:PTRansition" on page 953
- ":STATus:OPERation:MTESt[:EVENt]?" on page 954
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:MTESt:BIT<b>:NTRansition

N (see page 1292)

Command Syntax

```
:STATus:OPERation:MTESt:BIT<b>:NTRansition <bit_value>

<b> ::= 0-14; an integer in NR1 format

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

The :STATus:OPERation:MTESt:BIT<b>:NTRansition command sets a particular bit in the Negative Transition filter that identifies a Condition register bit on which a transition from 1 to 0 or TRUE to FALSE will be latched into the corresponding Event register bit.

Query Syntax

```
:STATus:OPERation:MTESt:BIT<b>:NTRansition?
```

The :STATus:OPERation:MTESt:BIT<b>:NTRansition? query returns the value of the specified Negative Transition filter bit.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:OPERation:MTESt:BIT<b>:CONDition?" on page 945
- ":STATus:OPERation:MTESt:BIT<b>:ENABle" on page 946
- ":STATus:OPERation:MTESt:BIT<b>:PTRansition" on page 948
- ":STATus:OPERation:MTESt:BIT<b>[:EVENt]?" on page 949
- ":STATus:OPERation:MTESt:CONDition?" on page 950
- ":STATus:OPERation:MTESt:ENABle" on page 951
- ":STATus:OPERation:MTESt:NTRansition" on page 952
- ":STATus:OPERation:MTESt:PTRansition" on page 953
- ":STATus:OPERation:MTESt[:EVENt]?" on page 954
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:MTESt:BIT<b>:PTRansition

N (see page 1292)

Command Syntax

```
:STATus:OPERation:MTESt:BIT<b>:PTRansition <bit_value>

<b> ::= 0-14; an integer in NR1 format

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

The :STATus:OPERation:MTESt:BIT<b>:PTRansition command sets a particular bit in the Positive Transition filter that identifies a Condition register bit on which a transition from 0 to 1 or FALSE to TRUE will be latched into the corresponding Event register bit.

Query Syntax

```
:STATus:OPERation:MTESt:BIT<b>:PTRansition?
```

The :STATus:OPERation:MTESt:BIT<b>:PTRansition? query returns the value of the specified Positive Transition filter bit.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:OPERation:MTESt:BIT<b>:CONDition?" on page 945
- ":STATus:OPERation:MTESt:BIT<b>:ENABle" on page 946
- ":STATus:OPERation:MTESt:BIT<b>:NTRansition" on page 947
- ":STATus:OPERation:MTESt:BIT<b>[:EVENt]?" on page 949
- ":STATus:OPERation:MTESt:CONDition?" on page 950
- ":STATus:OPERation:MTESt:ENABle" on page 951
- ":STATus:OPERation:MTESt:NTRansition" on page 952
- ":STATus:OPERation:MTESt:PTRansition" on page 953
- ":STATus:OPERation:MTESt[:EVENt]?" on page 954
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:MTESt:BIT<b>[:EVENt]?

(see page 1292)

Query Syntax

```
:STATus:OPERation:MTESt:BIT<b>[:EVENt]?

<b> ::= 0-14; an integer in NR1 format
```

The :STATus:OPERation:MTESt:BIT<b>[:EVENt]? query returns the value of the specified Event register bit and clears the bit.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:OPERation:MTESt:BIT<b>:CONDition?" on page 945
- ":STATus:OPERation:MTESt:BIT<b>:ENABle" on page 946
- ":STATus:OPERation:MTESt:BIT<b>:NTRansition" on page 947
- ":STATus:OPERation:MTESt:BIT<b>:PTRansition" on page 948
- ":STATus:OPERation:MTESt:CONDition?" on page 950
- ":STATus:OPERation:MTESt:ENABle" on page 951
- ":STATus:OPERation:MTESt:NTRansition" on page 952
- ":STATus:OPERation:MTESt:PTRansition" on page 953
- ":STATus:OPERation:MTESt[:EVENt]?" on page 954
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:MTESt:CONDition?

(see page 1292)

Query Syntax `:STATus:OPERation:MTESt:CONDition?`

The :STATus:OPERation:MTESt:CONDition? query returns the decimal value of the sum of the bits in the Condition register.

The Condition register continuously monitors status. Reading the Condition register does not affect its contents.

Return Format `<sum_of_bits><NL>`

`<sum_of_bits> ::= 0-32767; an integer in NR1 format`

See Also

- ":STATus:OPERation:MTESt:BIT<b>:CONDition?" on page 945
- ":STATus:OPERation:MTESt:BIT<b>:ENABle" on page 946
- ":STATus:OPERation:MTESt:BIT<b>:NTRansition" on page 947
- ":STATus:OPERation:MTESt:BIT<b>:PTRansition" on page 948
- ":STATus:OPERation:MTESt:BIT<b>[:EVENt]?" on page 949
- ":STATus:OPERation:MTESt:ENABle" on page 951
- ":STATus:OPERation:MTESt:NTRansition" on page 952
- ":STATus:OPERation:MTESt:PTRansition" on page 953
- ":STATus:OPERation:MTESt[:EVENt]?" on page 954
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:MTESt:ENABle

N (see page 1292)

Command Syntax

```
:STATus:OPERation:MTESt:ENABle <sum_of_bits>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

The :STATus:OPERation:MTESt:ENABle command sets bits in the Enable register that identify which Event register bits are ORed to create the Summary Message bit that is sent to the parent status register.

Query Syntax

```
:STATus:OPERation:MTESt:ENABle?
```

The :STATus:OPERation:MTESt:ENABle? query returns the decimal value of the sum of the bits in the Enable register.

Return Format

```
<sum_of_bits><NL>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

See Also

- ":STATus:OPERation:MTESt:BIT<b>:CONDition?" on page 945
- ":STATus:OPERation:MTESt:BIT<b>:ENABle" on page 946
- ":STATus:OPERation:MTESt:BIT<b>:NTRansition" on page 947
- ":STATus:OPERation:MTESt:BIT<b>:PTRansition" on page 948
- ":STATus:OPERation:MTESt:BIT<b>[:EVENt]?" on page 949
- ":STATus:OPERation:MTESt:CONDition?" on page 950
- ":STATus:OPERation:MTESt:NTRansition" on page 952
- ":STATus:OPERation:MTESt:PTRansition" on page 953
- ":STATus:OPERation:MTESt[:EVENt]?" on page 954
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:MTESt:NTRansition

(see page 1292)

Command Syntax

```
:STATus:OPERation:MTESt:NTRansition <sum_of_bits>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

The :STATus:OPERation:MTESt:NTRansition command sets bits in the Negative Transition filter that identify the Condition register bits on which transitions from 1 to 0 or TRUE to FALSE will be latched into the Event register.

Query Syntax

```
:STATus:OPERation:MTESt:NTRansition?
```

The :STATus:OPERation:MTESt:NTRansition? query returns the decimal value of the sum of the bits in the Negative Transition filter.

Return Format

```
<sum_of_bits><NL>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

See Also

- ":STATus:OPERation:MTESt:BIT<b>:CONDition?" on page 945
- ":STATus:OPERation:MTESt:BIT<b>:ENABle" on page 946
- ":STATus:OPERation:MTESt:BIT<b>:NTRansition" on page 947
- ":STATus:OPERation:MTESt:BIT<b>:PTRansition" on page 948
- ":STATus:OPERation:MTESt:BIT<b>[:EVENt]?" on page 949
- ":STATus:OPERation:MTESt:CONDition?" on page 950
- ":STATus:OPERation:MTESt:ENABle" on page 951
- ":STATus:OPERation:MTESt:PTRansition" on page 953
- ":STATus:OPERation:MTESt[:EVENt]?" on page 954
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:MTESt:PTRansition

 (see page 1292)

Command Syntax

```
:STATus:OPERation:MTESt:PTRansition <sum_of_bits>

<sum_of_bits> ::= 0-32767; an integer in NR1 format

<b> ::= 0-14; an integer in NR1 format
```

The :STATus:OPERation:MTESt:PTRansition command sets bits in the Positive Transition filter that identify the Condition register bits on which transitions from 0 to 1 or FALSE to TRUE will be latched into the Event register.

Query Syntax

```
:STATus:OPERation:MTESt:PTRansition?
```

The :STATus:OPERation:MTESt:PTRansition? query returns the decimal value of the sum of the bits in the Positive Transition filter.

Return Format

```
<sum_of_bits><NL>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

See Also

- ":STATus:OPERation:MTESt:BIT<b>:CONDition?" on page 945
- ":STATus:OPERation:MTESt:BIT<b>:ENABle" on page 946
- ":STATus:OPERation:MTESt:BIT<b>:NTRansition" on page 947
- ":STATus:OPERation:MTESt:BIT<b>:PTRansition" on page 948
- ":STATus:OPERation:MTESt:BIT<b>[:EVENt]?" on page 949
- ":STATus:OPERation:MTESt:CONDition?" on page 950
- ":STATus:OPERation:MTESt:ENABle" on page 951
- ":STATus:OPERation:MTESt:NTRansition" on page 952
- ":STATus:OPERation:MTESt[:EVENt]?" on page 954
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:MTESt[:EVENt]?

N (see page 1292)

Query Syntax `:STATus:OPERation:MTESt[:EVENt]?`

The :STATus:OPERation:MTESt[:EVENt]? query returns the decimal value of the sum of the bits in the Event register and clears the register.

The contents of the Event register show the events that have occurred since the last time the register was read.

Return Format `<sum_of_bits><NL>`

`<sum_of_bits> ::= 0-32767; an integer in NR1 format`

See Also

- ":STATus:OPERation:MTESt:BIT<b>:CONDition?" on page 945
- ":STATus:OPERation:MTESt:BIT<b>:ENABle" on page 946
- ":STATus:OPERation:MTESt:BIT<b>:NTRansition" on page 947
- ":STATus:OPERation:MTESt:BIT<b>:PTRansition" on page 948
- ":STATus:OPERation:MTESt:BIT<b>[:EVENt]?" on page 949
- ":STATus:OPERation:MTESt:CONDition?" on page 950
- ":STATus:OPERation:MTESt:ENABle" on page 951
- ":STATus:OPERation:MTESt:NTRansition" on page 952
- ":STATus:OPERation:MTESt:PTRansition" on page 953
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:OVERload Commands

The :STATus:OPERation:OVERload commands control the Overload Operation Register.

Table 122 :STATus:OPERation:OVERload Commands Summary

Command	Query	Options and Query Returns
n/a	:STATus:OPERation:OVERload:BIT<b>:CONDition? (see page 958)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:OVERload:BIT<b>:ENABle <bit_value> (see page 959)	:STATus:OPERation:OVERload:BIT<b>:ENABle? (see page 959)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:OVERload:BIT<b>:NTRansition <bit_value> (see page 960)	:STATus:OPERation:OVERload:BIT<b>:NTRansition? (see page 960)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:OVERload:BIT<b>:PTRansition <bit_value> (see page 961)	:STATus:OPERation:OVERload:BIT<b>:PTRansition? (see page 961)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation:OVERload:BIT<b>[:EVENt]? (see page 962)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation:OVERload:CONDition? (see page 963)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:OVERload:ENABle <sum_of_bits> (see page 964)	:STATus:OPERation:OVERload:ENABle? (see page 964)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:OVERload:NTRansition <sum_of_bits> (see page 965)	:STATus:OPERation:OVERload:NTRansition? (see page 965)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format

Table 122 :STATus:OPERation:OVERload Commands Summary (continued)

Command	Query	Options and Query Returns
:STATus:OPERation:OVERload:PTRansition <sum_of_bits> (see page 966)	:STATus:OPERation:OVERload:PTRansition? (see page 966)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation:OVERload[:EVENt]? (see page 967)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format

Overload Operation Register

To see where the Overload Operation Register appears in the status register hierarchy, see Chapter 41, “Status Reporting,” starting on page 1253.

To unserstand the SCPI status register model, see "Introduction to :STATus Commands" on page 883.

The following bits appear in the Overload Operation Register.

Table 123 Overload Operation Register Bits

Bit	Name	Description	When Set (1 = High = True), Indicates:	From:
15	---	---	(Not used, always zero.)	---
14	---	---	(Not used.)	---
13	---	External Trigger Fault	Fault has occurred on the External Trigger input.	---
12	---	Channel Probe Fault Summary	Enabled event in the Probe Fault Register has occurred.	Probe Fault Register
11	---	---	(Not used.)	---
10	---	---	(Not used.)	---
9	---	---	(Not used.)	---
8	EXT TRIG OVLD	External Trigger Overload	Overload has occurred on the External Trigger input.	---
7	CH8 OVLD	Channel 8 Overload.	Overload has occurred on Channel 8 input.	---
6	CH7 OVLD	Channel 7 Overload.	Overload has occurred on Channel 7 input.	---
5	CH6 OVLD	Channel 6 Overload.	Overload has occurred on Channel 6 input.	---
4	CH5 OVLD	Channel 5 Overload.	Overload has occurred on Channel 5 input.	---
3	CH4 OVLD	Channel 4 Overload.	Overload has occurred on Channel 4 input.	---
2	CH3 OVLD	Channel 3 Overload.	Overload has occurred on Channel 3 input.	---
1	CH2 OVLD	Channel 2 Overload.	Overload has occurred on Channel 2 input.	---
0	CH1 OVLD	Channel 1 Overload.	Overload has occurred on Channel 1 input.	---

When enabled, the summary message for the Overload Operation Register goes to Operation Status Register bit 11 (OVLD). See ":STATus:OPERation Commands" on page 888.

:STATus:OPERation:OVERload:BIT<b>:CONDition?

(see page 1292)

Query Syntax

```
:STATus:OPERation:OVERload:BIT<b>:CONDition?

<b> ::= 0-14; an integer in NR1 format
```

The :STATus:OPERation:OVERload:BIT<b>:CONDition? query returns the value of a particular bit in the Condition register.

The Condition register continuously monitors status. Reading a Condition register bit does not affect its contents.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:OPERation:OVERload:BIT<b>:ENABle" on page 959
- ":STATus:OPERation:OVERload:BIT<b>:NTRansition" on page 960
- ":STATus:OPERation:OVERload:BIT<b>:PTRansition" on page 961
- ":STATus:OPERation:OVERload:BIT<b>[:EVENt]?" on page 962
- ":STATus:OPERation:OVERload:CONDition?" on page 963
- ":STATus:OPERation:OVERload:ENABle" on page 964
- ":STATus:OPERation:OVERload:NTRansition" on page 965
- ":STATus:OPERation:OVERload:PTRansition" on page 966
- ":STATus:OPERation:OVERload[:EVENt]?" on page 967
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:OVERload:BIT<b>:ENABle

 (see page 1292)

Command Syntax

```
:STATus:OPERation:OVERload:BIT<b>:ENABle <bit_value>

<b> ::= 0-14; an integer in NR1 format

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

The :STATus:OPERation:OVERload:BIT<b>:ENABle command sets a particular bit in the Enable register that identifies a corresponding Event register bit that will be ORed with other identified Event register bits to create the Summary Message bit that is sent to the parent status register.

Query Syntax

```
:STATus:OPERation:OVERload:BIT<b>:ENABle?
```

The :STATus:OPERation:OVERload:BIT<b>:ENABle? query returns the value of the specified Enable register bit.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:OPERation:OVERload:BIT<b>:CONDition?" on page 958
- ":STATus:OPERation:OVERload:BIT<b>:NTRansition" on page 960
- ":STATus:OPERation:OVERload:BIT<b>:PTRansition" on page 961
- ":STATus:OPERation:OVERload:BIT<b>[:EVENt]?" on page 962
- ":STATus:OPERation:OVERload:CONDition?" on page 963
- ":STATus:OPERation:OVERload:ENABle" on page 964
- ":STATus:OPERation:OVERload:NTRansition" on page 965
- ":STATus:OPERation:OVERload:PTRansition" on page 966
- ":STATus:OPERation:OVERload[:EVENt]?" on page 967
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:OVERload:BIT<b>:NTRansition

(see page 1292)

Command Syntax

```
:STATus:OPERation:OVERload:BIT<b>:NTRansition <bit_value>

<b> ::= 0-14; an integer in NR1 format

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

The :STATus:OPERation:OVERload:BIT<b>:NTRansition command sets a particular bit in the Negative Transition filter that identifies a Condition register bit on which a transition from 1 to 0 or TRUE to FALSE will be latched into the corresponding Event register bit.

Query Syntax

```
:STATus:OPERation:OVERload:BIT<b>:NTRansition?
```

The :STATus:OPERation:OVERload:BIT<b>:NTRansition? query returns the value of the specified Negative Transition filter bit.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:OPERation:OVERload:BIT<b>:CONDition?" on page 958
- ":STATus:OPERation:OVERload:BIT<b>:ENABle" on page 959
- ":STATus:OPERation:OVERload:BIT<b>:PTRansition" on page 961
- ":STATus:OPERation:OVERload:BIT<b>[:EVENt]?" on page 962
- ":STATus:OPERation:OVERload:CONDition?" on page 963
- ":STATus:OPERation:OVERload:ENABle" on page 964
- ":STATus:OPERation:OVERload:NTRansition" on page 965
- ":STATus:OPERation:OVERload:PTRansition" on page 966
- ":STATus:OPERation:OVERload[:EVENt]?" on page 967
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:OVERload:BIT<b>:PTRansition

 (see page 1292)

Command Syntax

```
:STATus:OPERation:OVERload:BIT<b>:PTRansition <bit_value>

<b> ::= 0-14; an integer in NR1 format

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

The :STATus:OPERation:OVERload:BIT<b>:PTRansition command sets a particular bit in the Positive Transition filter that identifies a Condition register bit on which a transition from 0 to 1 or FALSE to TRUE will be latched into the corresponding Event register bit.

Query Syntax

```
:STATus:OPERation:OVERload:BIT<b>:PTRansition?
```

The :STATus:OPERation:OVERload:BIT<b>:PTRansition? query returns the value of the specified Positive Transition filter bit.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:OPERation:OVERload:BIT<b>:CONDition?" on page 958
- ":STATus:OPERation:OVERload:BIT<b>:ENABle" on page 959
- ":STATus:OPERation:OVERload:BIT<b>:NTRansition" on page 960
- ":STATus:OPERation:OVERload:BIT<b>[:EVENt]?" on page 962
- ":STATus:OPERation:OVERload:CONDition?" on page 963
- ":STATus:OPERation:OVERload:ENABle" on page 964
- ":STATus:OPERation:OVERload:NTRansition" on page 965
- ":STATus:OPERation:OVERload:PTRansition" on page 966
- ":STATus:OPERation:OVERload[:EVENt]?" on page 967
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:OVERload:BIT<b>[:EVENt]?

(see page 1292)

Query Syntax

```
:STATus:OPERation:OVERload:BIT<b>[:EVENt]?

<b> ::= 0-14; an integer in NR1 format
```

The :STATus:OPERation:OVERload:BIT<b>[:EVENt]? query returns the value of the specified Event register bit and clears the bit.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:OPERation:OVERload:BIT<b>:CONDition?" on page 958
- ":STATus:OPERation:OVERload:BIT<b>:ENABle" on page 959
- ":STATus:OPERation:OVERload:BIT<b>:NTRansition" on page 960
- ":STATus:OPERation:OVERload:BIT<b>:PTRansition" on page 961
- ":STATus:OPERation:OVERload:CONDition?" on page 963
- ":STATus:OPERation:OVERload:ENABle" on page 964
- ":STATus:OPERation:OVERload:NTRansition" on page 965
- ":STATus:OPERation:OVERload:PTRansition" on page 966
- ":STATus:OPERation:OVERload[:EVENt]?" on page 967
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:OVERload:CONDition?

N (see page 1292)

Query Syntax `:STATus:OPERation:OVERload:CONDition?`

The :STATus:OPERation:OVERload:CONDition? query returns the decimal value of the sum of the bits in the Condition register.

The Condition register continuously monitors status. Reading the Condition register does not affect its contents.

Return Format `<sum_of_bits><NL>`

`<sum_of_bits> ::= 0-32767; an integer in NR1 format`

See Also

- ":STATus:OPERation:OVERload:BIT<b>:CONDition?" on page 958
- ":STATus:OPERation:OVERload:BIT<b>:ENABle" on page 959
- ":STATus:OPERation:OVERload:BIT<b>:NTRansition" on page 960
- ":STATus:OPERation:OVERload:BIT<b>:PTRansition" on page 961
- ":STATus:OPERation:OVERload:BIT<b>[:EVENt]?" on page 962
- ":STATus:OPERation:OVERload:ENABle" on page 964
- ":STATus:OPERation:OVERload:NTRansition" on page 965
- ":STATus:OPERation:OVERload:PTRansition" on page 966
- ":STATus:OPERation:OVERload[:EVENt]?" on page 967
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:OVERload:ENABle

N (see page 1292)

Command Syntax

```
:STATus:OPERation:OVERload:ENABle <sum_of_bits>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

The :STATus:OPERation:OVERload:ENABle command sets bits in the Enable register that identify which Event register bits are ORed to create the Summary Message bit that is sent to the parent status register.

Query Syntax

```
:STATus:OPERation:OVERload:ENABle?
```

The :STATus:OPERation:OVERload:ENABle? query returns the decimal value of the sum of the bits in the Enable register.

Return Format

```
<sum_of_bits><NL>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

See Also

- ":STATus:OPERation:OVERload:BIT<b>:CONDition?" on page 958
- ":STATus:OPERation:OVERload:BIT<b>:ENABle" on page 959
- ":STATus:OPERation:OVERload:BIT<b>:NTRansition" on page 960
- ":STATus:OPERation:OVERload:BIT<b>:PTRansition" on page 961
- ":STATus:OPERation:OVERload:BIT<b>[:EVENt]?" on page 962
- ":STATus:OPERation:OVERload:CONDition?" on page 963
- ":STATus:OPERation:OVERload:NTRansition" on page 965
- ":STATus:OPERation:OVERload:PTRansition" on page 966
- ":STATus:OPERation:OVERload[:EVENt]?" on page 967
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:OVERload:NTRansition

 (see page 1292)

Command Syntax

```
:STATus:OPERation:OVERload:NTRansition <sum_of_bits>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

The :STATus:OPERation:OVERload:NTRansition command sets bits in the Negative Transition filter that identify the Condition register bits on which transitions from 1 to 0 or TRUE to FALSE will be latched into the Event register.

Query Syntax

```
:STATus:OPERation:OVERload:NTRansition?
```

The :STATus:OPERation:OVERload:NTRansition? query returns the decimal value of the sum of the bits in the Negative Transition filter.

Return Format

```
<sum_of_bits><NL>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

See Also

- ":STATus:OPERation:OVERload:BIT<b>:CONDition?" on page 958
- ":STATus:OPERation:OVERload:BIT<b>:ENABle" on page 959
- ":STATus:OPERation:OVERload:BIT<b>:NTRansition" on page 960
- ":STATus:OPERation:OVERload:BIT<b>:PTRansition" on page 961
- ":STATus:OPERation:OVERload:BIT<b>[:EVENt]?" on page 962
- ":STATus:OPERation:OVERload:CONDition?" on page 963
- ":STATus:OPERation:OVERload:ENABle" on page 964
- ":STATus:OPERation:OVERload:PTRansition" on page 966
- ":STATus:OPERation:OVERload[:EVENt]?" on page 967
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:OVERload:PTRansition

N (see page 1292)

Command Syntax

```
:STATus:OPERation:OVERload:PTRansition <sum_of_bits>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

The :STATus:OPERation:OVERload:PTRansition command sets bits in the Positive Transition filter that identify the Condition register bits on which transitions from 0 to 1 or FALSE to TRUE will be latched into the Event register.

Query Syntax

```
:STATus:OPERation:OVERload:PTRansition?
```

The :STATus:OPERation:OVERload:PTRansition? query returns the decimal value of the sum of the bits in the Positive Transition filter.

Return Format

```
<sum_of_bits><NL>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

See Also

- ":STATus:OPERation:OVERload:BIT<b>:CONDition?" on page 958
- ":STATus:OPERation:OVERload:BIT<b>:ENABle" on page 959
- ":STATus:OPERation:OVERload:BIT<b>:NTRansition" on page 960
- ":STATus:OPERation:OVERload:BIT<b>:PTRansition" on page 961
- ":STATus:OPERation:OVERload:BIT<b>[:EVENt]?" on page 962
- ":STATus:OPERation:OVERload:CONDition?" on page 963
- ":STATus:OPERation:OVERload:ENABle" on page 964
- ":STATus:OPERation:OVERload:NTRansition" on page 965
- ":STATus:OPERation:OVERload[:EVENt]?" on page 967
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:OVERload[:EVENt]?

N (see page 1292)

Query Syntax `:STATus:OPERation:OVERload[:EVENt]?`

The :STATus:OPERation:OVERload[:EVENt]? query returns the decimal value of the sum of the bits in the Event register and clears the register.

The contents of the Event register show the events that have occurred since the last time the register was read.

Return Format `<sum_of_bits><NL>`

`<sum_of_bits> ::= 0-32767; an integer in NR1 format`

See Also

- ":STATus:OPERation:OVERload:BIT<b>:CONDition?" on page 958
- ":STATus:OPERation:OVERload:BIT<b>:ENABle" on page 959
- ":STATus:OPERation:OVERload:BIT<b>:NTRansition" on page 960
- ":STATus:OPERation:OVERload:BIT<b>:PTRansition" on page 961
- ":STATus:OPERation:OVERload:BIT<b>[:EVENt]?" on page 962
- ":STATus:OPERation:OVERload:CONDition?" on page 963
- ":STATus:OPERation:OVERload:ENABle" on page 964
- ":STATus:OPERation:OVERload:NTRansition" on page 965
- ":STATus:OPERation:OVERload:PTRansition" on page 966
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:OVERload:PFAult Commands

The :STATus:OPERation:OVERload:PFAult commands control the Probe Fault Register.

Table 124 :STATus:OPERation:OVERload:PFAult Commands Summary

Command	Query	Options and Query Returns
n/a	:STATus:OPERation:OVERload:PFAult:BIT<b>:CONDition? (see page 971)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:OVERload:PFAult:BIT<b>:ENABle <bit_value> (see page 972)	:STATus:OPERation:OVERload:PFAult:BIT<b>:ENABle? (see page 972)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:OVERload:PFAult:BIT<b>:NTRansition <bit_value> (see page 973)	:STATus:OPERation:OVERload:PFAult:BIT<b>:NTRansition? (see page 973)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:OVERload:PFAult:BIT<b>:PTRansition <bit_value> (see page 974)	:STATus:OPERation:OVERload:PFAult:BIT<b>:PTRansition? (see page 974)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation:OVERload:PFAult:BIT<b>[:EVENt]? (see page 975)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation:OVERload:PFAult:CONDition? (see page 976)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:OVERload:PFAult:ENABle <sum_of_bits> (see page 977)	:STATus:OPERation:OVERload:PFAult:ENABle? (see page 977)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:OVERload:PFAult:NTRansition <sum_of_bits> (see page 978)	:STATus:OPERation:OVERload:PFAult:NTRansition? (see page 978)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format

Table 124 :STATus:OPERation:OVERload:PFAult Commands Summary (continued)

Command	Query	Options and Query Returns
:STATus:OPERation:OVERload:PFAult:PTRansition <sum_of_bits> (see page 979)	:STATus:OPERation:OVERload:PFAult:PTRansition? (see page 979)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation:OVERload:PFAult[:EVENt]? (see page 980)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format

Probe Fault Register

To see where the Probe Fault Register appears in the status register hierarchy, see Chapter 41, “Status Reporting,” starting on page 1253.

To unserstand the SCPI status register model, see "Introduction to :STATus Commands" on page 883.

The following bits appear in the Overload Operation Register.

Table 125 Probe Fault Register Bits

Bit	Name	Description	When Set (1 = High = True), Indicates:
15	---	---	(Not used, always zero.)
14	---	---	(Not used.)
13	---	---	(Not used.)
12	---	---	(Not used.)
11	---	---	(Not used.)
10	---	---	(Not used.)
9	---	---	(Not used.)
8	---	---	(Not used.)
7	CH8 PFA	Channel 8 Probe Fault.	Fault has occurred on Channel 8 input.
6	CH7 PFA	Channel 7 Probe Fault.	Fault has occurred on Channel 7 input.
5	CH6 PFA	Channel 6 Probe Fault.	Fault has occurred on Channel 6 input.
4	CH5 PFA	Channel 5 Probe Fault.	Fault has occurred on Channel 5 input.
3	CH4 PFA	Channel 4 Probe Fault.	Fault has occurred on Channel 4 input.
2	CH3 PFA	Channel 3 Probe Fault.	Fault has occurred on Channel 3 input.
1	CH2 PFA	Channel 2 Probe Fault.	Fault has occurred on Channel 2 input.
0	CH1 PFA	Channel 1 Probe Fault.	Fault has occurred on Channel 1 input.

When enabled, the summary message for the Probe Fault Register goes to Overload Operation Register bit 12 (Chanel Probe Fault Summary). See ":STATus:OPERation:OVERload Commands" on page 955.

:STATus:OPERation:OVERload:PFAult:BIT<b>:CONDition?

N (see page 1292)

Query Syntax

```
:STATus:OPERation:OVERload:PFAult:BIT<b>:CONDition?

<b> ::= 0-14; an integer in NR1 format
```

The :STATus:OPERation:OVERload:PFAult:BIT<b>:CONDition? query returns the value of a particular bit in the Condition register.

The Condition register continuously monitors status. Reading a Condition register bit does not affect its contents.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:OPERation:OVERload:PFAult:BIT<b>:ENABle" on page 972
- ":STATus:OPERation:OVERload:PFAult:BIT<b>:NTRansition" on page 973
- ":STATus:OPERation:OVERload:PFAult:BIT<b>:PTRansition" on page 974
- ":STATus:OPERation:OVERload:PFAult:BIT<b>[:EVENt]?" on page 975
- ":STATus:OPERation:OVERload:PFAult:CONDition?" on page 976
- ":STATus:OPERation:OVERload:PFAult:ENABle" on page 977
- ":STATus:OPERation:OVERload:PFAult:NTRansition" on page 978
- ":STATus:OPERation:OVERload:PFAult:PTRansition" on page 979
- ":STATus:OPERation:OVERload:PFAult[:EVENt]?" on page 980
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:OVERload:PFAult:BIT<b>:ENABle

N (see page 1292)

Command Syntax

```
:STATus:OPERation:OVERload:PFAult:BIT<b>:ENABle <bit_value>

<b> ::= 0-14; an integer in NR1 format

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

The :STATus:OPERation:OVERload:PFAult:BIT<b>:ENABle command sets a particular bit in the Enable register that identifies a corresponding Event register bit that will be ORed with other identified Event register bits to create the Summary Message bit that is sent to the parent status register.

Query Syntax

```
:STATus:OPERation:OVERload:PFAult:BIT<b>:ENABle?
```

The :STATus:OPERation:OVERload:PFAult:BIT<b>:ENABle? query returns the value of the specified Enable register bit.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:OPERation:OVERload:PFAult:BIT<b>:CONDition?" on page 971
- ":STATus:OPERation:OVERload:PFAult:BIT<b>:NTRansition" on page 973
- ":STATus:OPERation:OVERload:PFAult:BIT<b>:PTRansition" on page 974
- ":STATus:OPERation:OVERload:PFAult:BIT<b>[:EVENt]?" on page 975
- ":STATus:OPERation:OVERload:PFAult:CONDition?" on page 976
- ":STATus:OPERation:OVERload:PFAult:ENABle" on page 977
- ":STATus:OPERation:OVERload:PFAult:NTRansition" on page 978
- ":STATus:OPERation:OVERload:PFAult:PTRansition" on page 979
- ":STATus:OPERation:OVERload:PFAult[:EVENt]?" on page 980
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:OVERload:PFAult:BIT<b>:NTRansition

 (see page 1292)

Command Syntax

```
:STATus:OPERation:OVERload:PFAult:BIT<b>:NTRansition <bit_value>

<b> ::= 0-14; an integer in NR1 format

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

The :STATus:OPERation:OVERload:PFAult:BIT<b>:NTRansition command sets a particular bit in the Negative Transition filter that identifies a Condition register bit on which a transition from 1 to 0 or TRUE to FALSE will be latched into the corresponding Event register bit.

Query Syntax

```
:STATus:OPERation:OVERload:PFAult:BIT<b>:NTRansition?
```

The :STATus:OPERation:OVERload:PFAult:BIT<b>:NTRansition? query returns the value of the specified Negative Transition filter bit.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:OPERation:OVERload:PFAult:BIT<b>:CONDition?" on page 971
- ":STATus:OPERation:OVERload:PFAult:BIT<b>:ENABle" on page 972
- ":STATus:OPERation:OVERload:PFAult:BIT<b>:PTRansition" on page 974
- ":STATus:OPERation:OVERload:PFAult:BIT<b>[:EVENt]?" on page 975
- ":STATus:OPERation:OVERload:PFAult:CONDition?" on page 976
- ":STATus:OPERation:OVERload:PFAult:ENABle" on page 977
- ":STATus:OPERation:OVERload:PFAult:NTRansition" on page 978
- ":STATus:OPERation:OVERload:PFAult:PTRansition" on page 979
- ":STATus:OPERation:OVERload:PFAult[:EVENt]?" on page 980
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:OVERload:PFAult:BIT<b>:PTRansition

 (see page 1292)

Command Syntax

```
:STATus:OPERation:OVERload:PFAult:BIT<b>:PTRansition <bit_value>

<b> ::= 0-14; an integer in NR1 format

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

The :STATus:OPERation:OVERload:PFAult:BIT<b>:PTRansition command sets a particular bit in the Positive Transition filter that identifies a Condition register bit on which a transition from 0 to 1 or FALSE to TRUE will be latched into the corresponding Event register bit.

Query Syntax

```
:STATus:OPERation:OVERload:PFAult:BIT<b>:PTRansition?
```

The :STATus:OPERation:OVERload:PFAult:BIT<b>:PTRansition? query returns the value of the specified Positive Transition filter bit.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:OPERation:OVERload:PFAult:BIT<b>:CONDition?" on page 971
- ":STATus:OPERation:OVERload:PFAult:BIT<b>:ENABle" on page 972
- ":STATus:OPERation:OVERload:PFAult:BIT<b>:NTRansition" on page 973
- ":STATus:OPERation:OVERload:PFAult:BIT<b>[:EVENt]?" on page 975
- ":STATus:OPERation:OVERload:PFAult:CONDition?" on page 976
- ":STATus:OPERation:OVERload:PFAult:ENABle" on page 977
- ":STATus:OPERation:OVERload:PFAult:NTRansition" on page 978
- ":STATus:OPERation:OVERload:PFAult:PTRansition" on page 979
- ":STATus:OPERation:OVERload:PFAult[:EVENt]?" on page 980
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:OVERload:PFAult:BIT<b>[:EVENt]?

(see page 1292)

Query Syntax

```
:STATus:OPERation:OVERload:PFAult:BIT<b>[:EVENt]?

<b> ::= 0-14; an integer in NR1 format
```

The :STATus:OPERation:OVERload:PFAult:BIT<b>[:EVENt]? query returns the value of the specified Event register bit and clears the bit.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:OPERation:OVERload:PFAult:BIT<b>:CONDition?" on page 971
- ":STATus:OPERation:OVERload:PFAult:BIT<b>:ENABle" on page 972
- ":STATus:OPERation:OVERload:PFAult:BIT<b>:NTRansition" on page 973
- ":STATus:OPERation:OVERload:PFAult:BIT<b>:PTRansition" on page 974
- ":STATus:OPERation:OVERload:PFAult:CONDition?" on page 976
- ":STATus:OPERation:OVERload:PFAult:ENABle" on page 977
- ":STATus:OPERation:OVERload:PFAult:NTRansition" on page 978
- ":STATus:OPERation:OVERload:PFAult:PTRansition" on page 979
- ":STATus:OPERation:OVERload:PFAult[:EVENt]?" on page 980
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:OVERload:PFAult:CONDition?

(see page 1292)

Query Syntax `:STATus:OPERation:OVERload:PFAult:CONDition?`

The :STATus:OPERation:OVERload:PFAult:CONDition? query returns the decimal value of the sum of the bits in the Condition register.

The Condition register continuously monitors status. Reading the Condition register does not affect its contents.

Return Format `<sum_of_bits><NL>`

`<sum_of_bits> ::= 0-32767; an integer in NR1 format`

See Also

- ":STATus:OPERation:OVERload:PFAult:BIT<b>:CONDition?" on page 971
- ":STATus:OPERation:OVERload:PFAult:BIT<b>:ENABle" on page 972
- ":STATus:OPERation:OVERload:PFAult:BIT<b>:NTRansition" on page 973
- ":STATus:OPERation:OVERload:PFAult:BIT<b>:PTRansition" on page 974
- ":STATus:OPERation:OVERload:PFAult:BIT<b>[:EVENt]?" on page 975
- ":STATus:OPERation:OVERload:PFAult:ENABle" on page 977
- ":STATus:OPERation:OVERload:PFAult:NTRansition" on page 978
- ":STATus:OPERation:OVERload:PFAult:PTRansition" on page 979
- ":STATus:OPERation:OVERload:PFAult[:EVENt]?" on page 980
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:OVERload:PFAult:ENABle

 (see page 1292)

Command Syntax

```
:STATus:OPERation:OVERload:PFAult:ENABle <sum_of_bits>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

The :STATus:OPERation:OVERload:PFAult:ENABle command sets bits in the Enable register that identify which Event register bits are ORed to create the Summary Message bit that is sent to the parent status register.

Query Syntax

```
:STATus:OPERation:OVERload:PFAult:ENABle?
```

The :STATus:OPERation:OVERload:PFAult:ENABle? query returns the decimal value of the sum of the bits in the Enable register.

Return Format

```
<sum_of_bits><NL>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

See Also

- ":STATus:OPERation:OVERload:PFAult:BIT<b>:CONDition?" on page 971
- ":STATus:OPERation:OVERload:PFAult:BIT<b>:ENABle" on page 972
- ":STATus:OPERation:OVERload:PFAult:BIT<b>:NTRansition" on page 973
- ":STATus:OPERation:OVERload:PFAult:BIT<b>:PTRansition" on page 974
- ":STATus:OPERation:OVERload:PFAult:BIT<b>[:EVENt]?" on page 975
- ":STATus:OPERation:OVERload:PFAult:CONDition?" on page 976
- ":STATus:OPERation:OVERload:PFAult:NTRansition" on page 978
- ":STATus:OPERation:OVERload:PFAult:PTRansition" on page 979
- ":STATus:OPERation:OVERload:PFAult[:EVENt]?" on page 980
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:OVERload:PFAult:NTRansition

(see page 1292)

Command Syntax

```
:STATus:OPERation:OVERload:PFAult:NTRansition <sum_of_bits>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

The :STATus:OPERation:OVERload:PFAult:NTRansition command sets bits in the Negative Transition filter that identify the Condition register bits on which transitions from 1 to 0 or TRUE to FALSE will be latched into the Event register.

Query Syntax

```
:STATus:OPERation:OVERload:PFAult:NTRansition?
```

The :STATus:OPERation:OVERload:PFAult:NTRansition? query returns the decimal value of the sum of the bits in the Negative Transition filter.

Return Format

```
<sum_of_bits><NL>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

See Also

- ":STATus:OPERation:OVERload:PFAult:BIT<b>:CONDition?" on page 971
- ":STATus:OPERation:OVERload:PFAult:BIT<b>:ENABle" on page 972
- ":STATus:OPERation:OVERload:PFAult:BIT<b>:NTRansition" on page 973
- ":STATus:OPERation:OVERload:PFAult:BIT<b>:PTRansition" on page 974
- ":STATus:OPERation:OVERload:PFAult:BIT<b>[:EVENt]?" on page 975
- ":STATus:OPERation:OVERload:PFAult:CONDition?" on page 976
- ":STATus:OPERation:OVERload:PFAult:ENABle" on page 977
- ":STATus:OPERation:OVERload:PFAult:PTRansition" on page 979
- ":STATus:OPERation:OVERload:PFAult[:EVENt]?" on page 980
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:OVERload:PFAult:PTRansition

 (see page 1292)

Command Syntax

```
:STATus:OPERation:OVERload:PFAult:PTRansition <sum_of_bits>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

The :STATus:OPERation:OVERload:PFAult:PTRansition command sets bits in the Positive Transition filter that identify the Condition register bits on which transitions from 0 to 1 or FALSE to TRUE will be latched into the Event register.

Query Syntax

```
:STATus:OPERation:OVERload:PFAult:PTRansition?
```

The :STATus:OPERation:OVERload:PFAult:PTRansition? query returns the decimal value of the sum of the bits in the Positive Transition filter.

Return Format

```
<sum_of_bits><NL>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

See Also

- ":STATus:OPERation:OVERload:PFAult:BIT<b>:CONDition?" on page 971
- ":STATus:OPERation:OVERload:PFAult:BIT<b>:ENABle" on page 972
- ":STATus:OPERation:OVERload:PFAult:BIT<b>:NTRansition" on page 973
- ":STATus:OPERation:OVERload:PFAult:BIT<b>:PTRansition" on page 974
- ":STATus:OPERation:OVERload:PFAult:BIT<b>[:EVENt]?" on page 975
- ":STATus:OPERation:OVERload:PFAult:CONDition?" on page 976
- ":STATus:OPERation:OVERload:PFAult:ENABle" on page 977
- ":STATus:OPERation:OVERload:PFAult:NTRansition" on page 978
- ":STATus:OPERation:OVERload:PFAult[:EVENt]?" on page 980
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:OVERload:PFAult[:EVENt]?

(see page 1292)

Query Syntax `:STATus:OPERation:OVERload:PFAult[:EVENt]?`

The :STATus:OPERation:OVERload:PFAult[:EVENt]? query returns the decimal value of the sum of the bits in the Event register and clears the register.

The contents of the Event register show the events that have occurred since the last time the register was read.

Return Format `<sum_of_bits><NL>`

`<sum_of_bits> ::= 0-32767; an integer in NR1 format`

See Also

- ":STATus:OPERation:OVERload:PFAult:BIT<b>:CONDition?" on page 971
- ":STATus:OPERation:OVERload:PFAult:BIT<b>:ENABle" on page 972
- ":STATus:OPERation:OVERload:PFAult:BIT<b>:NTRansition" on page 973
- ":STATus:OPERation:OVERload:PFAult:BIT<b>:PTRansition" on page 974
- ":STATus:OPERation:OVERload:PFAult:BIT<b>[:EVENt]?" on page 975
- ":STATus:OPERation:OVERload:PFAult:CONDition?" on page 976
- ":STATus:OPERation:OVERload:PFAult:ENABle" on page 977
- ":STATus:OPERation:OVERload:PFAult:NTRansition" on page 978
- ":STATus:OPERation:OVERload:PFAult:PTRansition" on page 979
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:POWer Commands

The :STATus:OPERation:POWer commands control the Power Operation Register.

Table 126 :STATus:OPERation:POWer Commands Summary

Command	Query	Options and Query Returns
n/a	:STATus:OPERation:POWer:BIT<b>:CONDition? (see page 984)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:POWer:BIT<b>:ENABle <bit_value> (see page 985)	:STATus:OPERation:POWer:BIT<b>:ENABle? (see page 985)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:POWer:BIT<b>:NTRansition <bit_value> (see page 986)	:STATus:OPERation:POWer:BIT<b>:NTRansition? (see page 986)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:POWer:BIT<b>:PTRansition <bit_value> (see page 987)	:STATus:OPERation:POWer:BIT<b>:PTRansition? (see page 987)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation:POWer:BIT<b>[:EVENt]? (see page 988)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation:POWer:CONDition? (see page 989)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:POWer:ENABle <sum_of_bits> (see page 990)	:STATus:OPERation:POWer:ENABle? (see page 990)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:OPERation:POWer:NTRansition <sum_of_bits> (see page 991)	:STATus:OPERation:POWer:NTRansition? (see page 991)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format

Table 126 :STATus:OPERation:POWer Commands Summary (continued)

Command	Query	Options and Query Returns
:STATus:OPERation:POWer:PTRansition <sum_of_bits> (see page 992)	:STATus:OPERation:POWer:PTRansition? (see page 992)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:OPERation:POWer[:EVENt]? (see page 993)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format

Power Operation Register

To see where the Power Operation Register appears in the status register hierarchy, see Chapter 41, “Status Reporting,” starting on page 1253.

To unserstand the SCPI status register model, see "Introduction to :STATus Commands" on page 883.

The following bits appear in the Power Operation Register.

Table 127 Power Operation Register Bits

Bit	Name	Description	When Set (1 = High = True), Indicates:
15	---	---	(Not used, always zero.)
14	---	---	(Not used.)
13	---	---	(Not used.)
12	---	---	(Not used.)
11	---	---	(Not used.)
10	---	---	(Not used.)
9	---	---	(Not used.)
8	---	---	(Not used.)
7	---	---	(Not used.)
6	---	---	(Not used.)
5	---	---	(Not used.)
4	---	---	(Not used.)
3	---	---	(Not used.)
2	Deskew Complete	Deskew Complete	A power application deskew operation is complete.
1	Apply Complete	Apply Complete	A power application apply operation is complete.
0	Setup Complete	Setup Complete	A power application setup operation is complete.

When enabled, the summary message for the Power Operation Register goes to Operation Status Register bit 7 (Power). See ":STATus:OPERation Commands" on page 888.

:STATus:OPERation:POWer:BIT<b>:CONDition?

N (see page 1292)

Query Syntax

```
:STATus:OPERation:POWer:BIT<b>:CONDition?

<b> ::= 0-14; an integer in NR1 format
```

The :STATus:OPERation:POWer:BIT<b>:CONDition? query returns the value of a particular bit in the Condition register.

The Condition register continuously monitors status. Reading a Condition register bit does not affect its contents.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:OPERation:POWer:BIT<b>:ENABle" on page 985
- ":STATus:OPERation:POWer:BIT<b>:NTRansition" on page 986
- ":STATus:OPERation:POWer:BIT<b>:PTRansition" on page 987
- ":STATus:OPERation:POWer:BIT<b>[:EVENt]?" on page 988
- ":STATus:OPERation:POWer:CONDition?" on page 989
- ":STATus:OPERation:POWer:ENABle" on page 990
- ":STATus:OPERation:POWer:NTRansition" on page 991
- ":STATus:OPERation:POWer:PTRansition" on page 992
- ":STATus:OPERation:POWer[:EVENt]?" on page 993
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:POWer:BIT<b>:ENABle

 (see page 1292)

Command Syntax

```
:STATus:OPERation:POWer:BIT<b>:ENABle <bit_value>

<b> ::= 0-14; an integer in NR1 format

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

The :STATus:OPERation:POWer:BIT<b>:ENABle command sets a particular bit in the Enable register that identifies a corresponding Event register bit that will be ORed with other identified Event register bits to create the Summary Message bit that is sent to the parent status register.

Query Syntax

```
:STATus:OPERation:POWer:BIT<b>:ENABle?
```

The :STATus:OPERation:POWer:BIT<b>:ENABle? query returns the value of the specified Enable register bit.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:OPERation:POWer:BIT<b>:CONDition?" on page 984
- ":STATus:OPERation:POWer:BIT<b>:NTRansition" on page 986
- ":STATus:OPERation:POWer:BIT<b>:PTRansition" on page 987
- ":STATus:OPERation:POWer:BIT<b>[:EVENt]?" on page 988
- ":STATus:OPERation:POWer:CONDition?" on page 989
- ":STATus:OPERation:POWer:ENABle" on page 990
- ":STATus:OPERation:POWer:NTRansition" on page 991
- ":STATus:OPERation:POWer:PTRansition" on page 992
- ":STATus:OPERation:POWer[:EVENt]?" on page 993
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:POWer:BIT<b>:NTRansition

N (see page 1292)

Command Syntax

```
:STATus:OPERation:POWer:BIT<b>:NTRansition <bit_value>

<b> ::= 0-14; an integer in NR1 format

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

The :STATus:OPERation:POWer:BIT<b>:NTRansition command sets a particular bit in the Negative Transition filter that identifies a Condition register bit on which a transition from 1 to 0 or TRUE to FALSE will be latched into the corresponding Event register bit.

Query Syntax

```
:STATus:OPERation:POWer:BIT<b>:NTRansition?
```

The :STATus:OPERation:POWer:BIT<b>:NTRansition? query returns the value of the specified Negative Transition filter bit.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:OPERation:POWer:BIT<b>:CONDition?" on page 984
- ":STATus:OPERation:POWer:BIT<b>:ENABle" on page 985
- ":STATus:OPERation:POWer:BIT<b>:PTRansition" on page 987
- ":STATus:OPERation:POWer:BIT<b>[:EVENt]?" on page 988
- ":STATus:OPERation:POWer:CONDition?" on page 989
- ":STATus:OPERation:POWer:ENABle" on page 990
- ":STATus:OPERation:POWer:NTRansition" on page 991
- ":STATus:OPERation:POWer:PTRansition" on page 992
- ":STATus:OPERation:POWer[:EVENt]?" on page 993
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:POWer:BIT<b>:PTRansition

N (see page 1292)

Command Syntax

```
:STATus:OPERation:POWer:BIT<b>:PTRansition <bit_value>

<b> ::= 0-14; an integer in NR1 format

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

The :STATus:OPERation:POWer:BIT<b>:PTRansition command sets a particular bit in the Positive Transition filter that identifies a Condition register bit on which a transition from 0 to 1 or FALSE to TRUE will be latched into the corresponding Event register bit.

Query Syntax

```
:STATus:OPERation:POWer:BIT<b>:PTRansition?
```

The :STATus:OPERation:POWer:BIT<b>:PTRansition? query returns the value of the specified Positive Transition filter bit.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:OPERation:POWer:BIT<b>:CONDition?" on page 984
- ":STATus:OPERation:POWer:BIT<b>:ENABle" on page 985
- ":STATus:OPERation:POWer:BIT<b>:NTRansition" on page 986
- ":STATus:OPERation:POWer:BIT<b>[:EVENt]?" on page 988
- ":STATus:OPERation:POWer:CONDition?" on page 989
- ":STATus:OPERation:POWer:ENABle" on page 990
- ":STATus:OPERation:POWer:NTRansition" on page 991
- ":STATus:OPERation:POWer:PTRansition" on page 992
- ":STATus:OPERation:POWer[:EVENt]?" on page 993
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:POWer:BIT<b>[:EVENt]?

(see page 1292)

Query Syntax

```
:STATus:OPERation:POWer:BIT<b>[:EVENt]?

<b> ::= 0-14; an integer in NR1 format
```

The :STATus:OPERation:POWer:BIT<b>[:EVENt]? query returns the value of the specified Event register bit and clears the bit.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:OPERation:POWer:BIT<b>:CONDition?" on page 984
- ":STATus:OPERation:POWer:BIT<b>:ENABle" on page 985
- ":STATus:OPERation:POWer:BIT<b>:NTRansition" on page 986
- ":STATus:OPERation:POWer:BIT<b>:PTRansition" on page 987
- ":STATus:OPERation:POWer:CONDition?" on page 989
- ":STATus:OPERation:POWer:ENABle" on page 990
- ":STATus:OPERation:POWer:NTRansition" on page 991
- ":STATus:OPERation:POWer:PTRansition" on page 992
- ":STATus:OPERation:POWer[:EVENt]?" on page 993
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:POWer:CONDition?

N (see page 1292)

Query Syntax

```
:STATus:OPERation:POWer:CONDition?
```

The :STATus:OPERation:POWer:CONDition? query returns the decimal value of the sum of the bits in the Condition register.

The Condition register continuously monitors status. Reading the Condition register does not affect its contents.

Return Format

```
<sum_of_bits><NL>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

See Also

- ":STATus:OPERation:POWer:BIT<b>:CONDition?" on page 984
- ":STATus:OPERation:POWer:BIT<b>:ENABle" on page 985
- ":STATus:OPERation:POWer:BIT<b>:NTRansition" on page 986
- ":STATus:OPERation:POWer:BIT<b>:PTRansition" on page 987
- ":STATus:OPERation:POWer:BIT<b>[:EVENt]?" on page 988
- ":STATus:OPERation:POWer:ENABle" on page 990
- ":STATus:OPERation:POWer:NTRansition" on page 991
- ":STATus:OPERation:POWer:PTRansition" on page 992
- ":STATus:OPERation:POWer[:EVENt]?" on page 993
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:POWer:ENABle

(see page 1292)

Command Syntax

```
:STATus:OPERation:POWer:ENABle <sum_of_bits>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

The :STATus:OPERation:POWer:ENABle command sets bits in the Enable register that identify which Event register bits are ORed to create the Summary Message bit that is sent to the parent status register.

Query Syntax

```
:STATus:OPERation:POWer:ENABle?
```

The :STATus:OPERation:POWer:ENABle? query returns the decimal value of the sum of the bits in the Enable register.

Return Format

```
<sum_of_bits><NL>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

See Also

- ":STATus:OPERation:POWer:BIT<b>:CONDition?" on page 984
- ":STATus:OPERation:POWer:BIT<b>:ENABle" on page 985
- ":STATus:OPERation:POWer:BIT<b>:NTRansition" on page 986
- ":STATus:OPERation:POWer:BIT<b>:PTRansition" on page 987
- ":STATus:OPERation:POWer:BIT<b>[:EVENt]?" on page 988
- ":STATus:OPERation:POWer:CONDition?" on page 989
- ":STATus:OPERation:POWer:NTRansition" on page 991
- ":STATus:OPERation:POWer:PTRansition" on page 992
- ":STATus:OPERation:POWer[:EVENt]?" on page 993
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:POWer:NTRansition

(see page 1292)

Command Syntax

```
:STATus:OPERation:POWer:NTRansition <sum_of_bits>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

The :STATus:OPERation:POWer:NTRansition command sets bits in the Negative Transition filter that identify the Condition register bits on which transitions from 1 to 0 or TRUE to FALSE will be latched into the Event register.

Query Syntax

```
:STATus:OPERation:POWer:NTRansition?
```

The :STATus:OPERation:POWer:NTRansition? query returns the decimal value of the sum of the bits in the Negative Transition filter.

Return Format

```
<sum_of_bits><NL>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

See Also

- ":STATus:OPERation:POWer:BIT<b>:CONDition?" on page 984
- ":STATus:OPERation:POWer:BIT<b>:ENABle" on page 985
- ":STATus:OPERation:POWer:BIT<b>:NTRansition" on page 986
- ":STATus:OPERation:POWer:BIT<b>:PTRansition" on page 987
- ":STATus:OPERation:POWer:BIT<b>[:EVENt]?" on page 988
- ":STATus:OPERation:POWer:CONDition?" on page 989
- ":STATus:OPERation:POWer:ENABle" on page 990
- ":STATus:OPERation:POWer:PTRansition" on page 992
- ":STATus:OPERation:POWer[:EVENt]?" on page 993
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:POWer:PTRansition

N (see page 1292)

Command Syntax

```
:STATus:OPERation:POWer:PTRansition <sum_of_bits>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

The :STATus:OPERation:POWer:PTRansition command sets bits in the Positive Transition filter that identify the Condition register bits on which transitions from 0 to 1 or FALSE to TRUE will be latched into the Event register.

Query Syntax

```
:STATus:OPERation:POWer:PTRansition?
```

The :STATus:OPERation:POWer:PTRansition? query returns the decimal value of the sum of the bits in the Positive Transition filter.

Return Format

```
<sum_of_bits><NL>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

See Also

- ":STATus:OPERation:POWer:BIT<b>:CONDition?" on page 984
- ":STATus:OPERation:POWer:BIT<b>:ENABle" on page 985
- ":STATus:OPERation:POWer:BIT<b>:NTRansition" on page 986
- ":STATus:OPERation:POWer:BIT<b>:PTRansition" on page 987
- ":STATus:OPERation:POWer:BIT<b>[:EVENt]?" on page 988
- ":STATus:OPERation:POWer:CONDition?" on page 989
- ":STATus:OPERation:POWer:ENABle" on page 990
- ":STATus:OPERation:POWer:NTRansition" on page 991
- ":STATus:OPERation:POWer[:EVENt]?" on page 993
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:OPERation:POWer[:EVENt]?

N (see page 1292)

Query Syntax `:STATus:OPERation:POWer[:EVENt]?`

The :STATus:OPERation:POWer[:EVENt]? query returns the decimal value of the sum of the bits in the Event register and clears the register.

The contents of the Event register show the events that have occurred since the last time the register was read.

Return Format `<sum_of_bits><NL>`

`<sum_of_bits> ::= 0-32767; an integer in NR1 format`

See Also

- ":STATus:OPERation:POWer:BIT<b>:CONDition?" on page 984
- ":STATus:OPERation:POWer:BIT<b>:ENABle" on page 985
- ":STATus:OPERation:POWer:BIT<b>:NTRansition" on page 986
- ":STATus:OPERation:POWer:BIT<b>:PTRansition" on page 987
- ":STATus:OPERation:POWer:BIT<b>[:EVENt]?" on page 988
- ":STATus:OPERation:POWer:CONDition?" on page 989
- ":STATus:OPERation:POWer:ENABle" on page 990
- ":STATus:OPERation:POWer:NTRansition" on page 991
- ":STATus:OPERation:POWer:PTRansition" on page 992
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:TRIGger Commands

The :STATus:TRIGger commands control the Trigger Event Register.

Table 128 :STATus:TRIGger Commands Summary

Command	Query	Options and Query Returns
n/a	:STATus:TRIGger:BIT<b>:CONDition? (see page 997)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:TRIGger:BIT<b>:ENABle <bit_value> (see page 998)	:STATus:TRIGger:BIT<b>:ENABle? (see page 998)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:TRIGger:BIT<b>:NTRansition <bit_value> (see page 999)	:STATus:TRIGger:BIT<b>:NTRansition? (see page 999)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:TRIGger:BIT<b>:PTRansition <bit_value> (see page 1000)	:STATus:TRIGger:BIT<b>:PTRansition? (see page 1000)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:TRIGger:BIT<b>[:EVENt]? (see page 1001)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:TRIGger:CONDition? (see page 1002)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:TRIGger:ENABle <sum_of_bits> (see page 1003)	:STATus:TRIGger:ENABle? (see page 1003)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:TRIGger:NTRansition <sum_of_bits> (see page 1004)	:STATus:TRIGger:NTRansition? (see page 1004)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format

Table 128 :STATus:TRIGger Commands Summary (continued)

Command	Query	Options and Query Returns
:STATus:TRIGger:PTRansition <sum_of_bits> (see page 1005)	:STATus:TRIGger:PTRansition? (see page 1005)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:TRIGger[:EVENt]? (see page 1006)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format

Trigger Event Register

To see where the Trigger Event Register appears in the status register hierarchy, see Chapter 41, “Status Reporting,” starting on page 1253.

To unserstand the SCPI status register model, see "Introduction to :STATus Commands" on page 883.

The following bits appear in the Trigger Event Register.

Table 129 Trigger Event Register Bits

Bit	Name	Description	When Set (1 = High = True), Indicates:
15	---	---	(Not used, always zero.)
14	---	---	(Not used.)
13	---	---	(Not used.)
12	---	---	(Not used.)
11	---	---	(Not used.)
10	---	---	(Not used.)
9	---	---	(Not used.)
8	---	---	(Not used.)
7	---	---	(Not used.)
6	---	---	(Not used.)
5	---	---	(Not used.)
4	---	---	(Not used.)
3	---	---	(Not used.)
2	---	---	(Not used.)
1	---	---	(Not used.)
0	TER	Trigger Event	A trigger has occurred.

When enabled, the summary message for the Trigger Event Register goes to Status Byte Register bit 0 (TRG). See "*CLS (Clear Status)" on page 184.

:STATus:TRIGger:BIT<b>:CONDition?

(see page 1292)

Query Syntax

```
:STATus:TRIGger:BIT<b>:CONDition?

<b> ::= 0-14; an integer in NR1 format
```

The :STATus:TRIGger:BIT<b>:CONDition? query returns the value of a particular bit in the Condition register.

The Condition register continuously monitors status. Reading a Condition register bit does not affect its contents.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:TRIGger:BIT<b>:ENABle" on page 998
- ":STATus:TRIGger:BIT<b>:NTRansition" on page 999
- ":STATus:TRIGger:BIT<b>:PTRansition" on page 1000
- ":STATus:TRIGger:BIT<b>[:EVENt]?" on page 1001
- ":STATus:TRIGger:CONDition?" on page 1002
- ":STATus:TRIGger:ENABle" on page 1003
- ":STATus:TRIGger:NTRansition" on page 1004
- ":STATus:TRIGger:PTRansition" on page 1005
- ":STATus:TRIGger[:EVENt]?" on page 1006
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:TRIGger:BIT<b>:ENABle

N (see page 1292)

Command Syntax

```
:STATus:TRIGger:BIT<b>:ENABle <bit_value>

<b> ::= 0-14; an integer in NR1 format

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

The :STATus:TRIGger:BIT<b>:ENABle command sets a particular bit in the Enable register that identifies a corresponding Event register bit that will be ORed with other identified Event register bits to create the Summary Message bit that is sent to the parent status register.

Query Syntax

```
:STATus:TRIGger:BIT<b>:ENABle?
```

The :STATus:TRIGger:BIT<b>:ENABle? query returns the value of the specified Enable register bit.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:TRIGger:BIT<b>:CONDition?" on page 997
- ":STATus:TRIGger:BIT<b>:NTRansition" on page 999
- ":STATus:TRIGger:BIT<b>:PTRansition" on page 1000
- ":STATus:TRIGger:BIT<b>[:EVENt]?" on page 1001
- ":STATus:TRIGger:CONDition?" on page 1002
- ":STATus:TRIGger:ENABle" on page 1003
- ":STATus:TRIGger:NTRansition" on page 1004
- ":STATus:TRIGger:PTRansition" on page 1005
- ":STATus:TRIGger[:EVENt]?" on page 1006
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:TRIGger:BIT<b>:NTRansition

N (see page 1292)

Command Syntax

```
:STATus:TRIGger:BIT<b>:NTRansition <bit_value>

<b> ::= 0-14; an integer in NR1 format

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

The :STATus:TRIGger:BIT<b>:NTRansition command sets a particular bit in the Negative Transition filter that identifies a Condition register bit on which a transition from 1 to 0 or TRUE to FALSE will be latched into the corresponding Event register bit.

Query Syntax

```
:STATus:TRIGger:BIT<b>:NTRansition?
```

The :STATus:TRIGger:BIT<b>:NTRansition? query returns the value of the specified Negative Transition filter bit.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:TRIGger:BIT<b>:CONDition?" on page 997
- ":STATus:TRIGger:BIT<b>:ENABle" on page 998
- ":STATus:TRIGger:BIT<b>:PTRansition" on page 1000
- ":STATus:TRIGger:BIT<b>[:EVENt]?" on page 1001
- ":STATus:TRIGger:CONDition?" on page 1002
- ":STATus:TRIGger:ENABle" on page 1003
- ":STATus:TRIGger:NTRansition" on page 1004
- ":STATus:TRIGger:PTRansition" on page 1005
- ":STATus:TRIGger[:EVENt]?" on page 1006
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:TRIGger:BIT<b>:PTRansition

(see page 1292)

Command Syntax

```
:STATus:TRIGger:BIT<b>:PTRansition <bit_value>

<b> ::= 0-14; an integer in NR1 format

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

The :STATus:TRIGger:BIT<b>:PTRansition command sets a particular bit in the Positive Transition filter that identifies a Condition register bit on which a transition from 0 to 1 or FALSE to TRUE will be latched into the corresponding Event register bit.

Query Syntax

```
:STATus:TRIGger:BIT<b>:PTRansition?
```

The :STATus:TRIGger:BIT<b>:PTRansition? query returns the value of the specified Positive Transition filter bit.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:TRIGger:BIT<b>:CONDition?" on page 997
- ":STATus:TRIGger:BIT<b>:ENABle" on page 998
- ":STATus:TRIGger:BIT<b>:NTRansition" on page 999
- ":STATus:TRIGger:BIT<b>[:EVENt]?" on page 1001
- ":STATus:TRIGger:CONDition?" on page 1002
- ":STATus:TRIGger:ENABle" on page 1003
- ":STATus:TRIGger:NTRansition" on page 1004
- ":STATus:TRIGger:PTRansition" on page 1005
- ":STATus:TRIGger[:EVENt]?" on page 1006
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:TRIGger:BIT<b>[:EVENt]?

(see page 1292)

Query Syntax

```
:STATus:TRIGger:BIT<b>[:EVENt]?

<b> ::= 0-14; an integer in NR1 format
```

The :STATus:TRIGger:BIT<b>[:EVENt]? query returns the value of the specified Event register bit and clears the bit.

NOTE The :TER? query is an alias for the :STATus:TRIGger:BIT0[:EVENt]? query.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:TRIGger:BIT<b>:CONDition?" on page 997
- ":STATus:TRIGger:BIT<b>:ENABle" on page 998
- ":STATus:TRIGger:BIT<b>:NTRansition" on page 999
- ":STATus:TRIGger:BIT<b>:PTRansition" on page 1000
- ":STATus:TRIGger:CONDition?" on page 1002
- ":STATus:TRIGger:ENABle" on page 1003
- ":STATus:TRIGger:NTRansition" on page 1004
- ":STATus:TRIGger:PTRansition" on page 1005
- ":STATus:TRIGger[:EVENt]?" on page 1006
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184
- ":TER? (Trigger Event Register)" on page 226

:STATus:TRIGger:CONDition?

(see page 1292)

Query Syntax `:STATus:TRIGger:CONDition?`

The :STATus:TRIGger:CONDition? query returns the decimal value of the sum of the bits in the Condition register.

The Condition register continuously monitors status. Reading the Condition register does not affect its contents.

Return Format `<sum_of_bits><NL>`

`<sum_of_bits> ::= 0-32767; an integer in NR1 format`

See Also

- ":STATus:TRIGger:BIT<b>:CONDition?" on page 997
- ":STATus:TRIGger:BIT<b>:ENABle" on page 998
- ":STATus:TRIGger:BIT<b>:NTRansition" on page 999
- ":STATus:TRIGger:BIT<b>:PTRansition" on page 1000
- ":STATus:TRIGger:BIT<b>[:EVENt]?" on page 1001
- ":STATus:TRIGger:ENABle" on page 1003
- ":STATus:TRIGger:NTRansition" on page 1004
- ":STATus:TRIGger:PTRansition" on page 1005
- ":STATus:TRIGger[:EVENt]?" on page 1006
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:TRIGger:ENABle

(see page 1292)

Command Syntax

```
:STATus:TRIGger:ENABle <sum_of_bits>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

The :STATus:TRIGger:ENABle command sets bits in the Enable register that identify which Event register bits are ORed to create the Summary Message bit that is sent to the parent status register.

Query Syntax

```
:STATus:TRIGger:ENABle?
```

The :STATus:TRIGger:ENABle? query returns the decimal value of the sum of the bits in the Enable register.

Return Format

```
<sum_of_bits><NL>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

See Also

- ":STATus:TRIGger:BIT<b>:CONDition?" on page 997
- ":STATus:TRIGger:BIT<b>:ENABle" on page 998
- ":STATus:TRIGger:BIT<b>:NTRansition" on page 999
- ":STATus:TRIGger:BIT<b>:PTRansition" on page 1000
- ":STATus:TRIGger:BIT<b>[:EVENt]?" on page 1001
- ":STATus:TRIGger:CONDition?" on page 1002
- ":STATus:TRIGger:NTRansition" on page 1004
- ":STATus:TRIGger:PTRansition" on page 1005
- ":STATus:TRIGger[:EVENt]?" on page 1006
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:TRIGger:NTRansition

(see page 1292)

Command Syntax

```
:STATus:TRIGger:NTRansition <sum_of_bits>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

The :STATus:TRIGger:NTRansition command sets bits in the Negative Transition filter that identify the Condition register bits on which transitions from 1 to 0 or TRUE to FALSE will be latched into the Event register.

Query Syntax

```
:STATus:TRIGger:NTRansition?
```

The :STATus:TRIGger:NTRansition? query returns the decimal value of the sum of the bits in the Negative Transition filter.

Return Format

```
<sum_of_bits><NL>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

See Also

- ":STATus:TRIGger:BIT<b>:CONDition?" on page 997
- ":STATus:TRIGger:BIT<b>:ENABle" on page 998
- ":STATus:TRIGger:BIT<b>:NTRansition" on page 999
- ":STATus:TRIGger:BIT<b>:PTRansition" on page 1000
- ":STATus:TRIGger:BIT<b>[:EVENt]?" on page 1001
- ":STATus:TRIGger:CONDition?" on page 1002
- ":STATus:TRIGger:ENABle" on page 1003
- ":STATus:TRIGger:PTRansition" on page 1005
- ":STATus:TRIGger[:EVENt]?" on page 1006
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:TRIGger:PTRansition

N (see page 1292)

Command Syntax

```
:STATus:TRIGger:PTRansition <sum_of_bits>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

The :STATus:TRIGger:PTRansition command sets bits in the Positive Transition filter that identify the Condition register bits on which transitions from 0 to 1 or FALSE to TRUE will be latched into the Event register.

Query Syntax

```
:STATus:TRIGger:PTRansition?
```

The :STATus:TRIGger:PTRansition? query returns the decimal value of the sum of the bits in the Positive Transition filter.

Return Format

```
<sum_of_bits><NL>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

See Also

- ":STATus:TRIGger:BIT<b>:CONDition?" on page 997
- ":STATus:TRIGger:BIT<b>:ENABle" on page 998
- ":STATus:TRIGger:BIT<b>:NTRansition" on page 999
- ":STATus:TRIGger:BIT<b>:PTRansition" on page 1000
- ":STATus:TRIGger:BIT<b>[:EVENt]?" on page 1001
- ":STATus:TRIGger:CONDition?" on page 1002
- ":STATus:TRIGger:ENABle" on page 1003
- ":STATus:TRIGger:NTRansition" on page 1004
- ":STATus:TRIGger[:EVENt]?" on page 1006
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:TRIGger[:EVENt]?

(see page 1292)

Query Syntax

```
:STATus:TRIGger[:EVENt]?
```

The :STATus:TRIGger[:EVENt]? query returns the decimal value of the sum of the bits in the Event register and clears the register.

The contents of the Event register show the events that have occurred since the last time the register was read.

NOTE Because bit 0 is the only bit being used in the Trigger Event Register and the only one that can be latched into the Event register, the :TER? query is an alias for the :STATus:TRIGger[:EVENt]? query.

Return Format

```
<sum_of_bits><NL>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

See Also

- ":STATus:TRIGger:BIT<b>:CONDition?" on page 997
- ":STATus:TRIGger:BIT<b>:ENABle" on page 998
- ":STATus:TRIGger:BIT<b>:NTRansition" on page 999
- ":STATus:TRIGger:BIT<b>:PTRansition" on page 1000
- ":STATus:TRIGger:BIT<b>[:EVENt]?" on page 1001
- ":STATus:TRIGger:CONDition?" on page 1002
- ":STATus:TRIGger:ENABle" on page 1003
- ":STATus:TRIGger:NTRansition" on page 1004
- ":STATus:TRIGger:PTRansition" on page 1005
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184
- ":TER? (Trigger Event Register)" on page 226

:STATus:USER Commands

The :STATus:USER commands control the User Event Register.

Table 130 :STATus:USER Commands Summary

Command	Query	Options and Query Returns
n/a	:STATus:USER:BIT<b>:CONDition? (see page 1010)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:USER:BIT<b>:ENABle <bit_value> (see page 1011)	:STATus:USER:BIT<b>:ENABle? (see page 1011)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:USER:BIT<b>:NTRansition <bit_value> (see page 1012)	:STATus:USER:BIT<b>:NTRansition? (see page 1012)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
:STATus:USER:BIT<b>:PTRansition <bit_value> (see page 1013)	:STATus:USER:BIT<b>:PTRansition? (see page 1013)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:USER:BIT<b>[:EVENt]? (see page 1014)	<bit_value> ::= {0 | 1|} <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:USER:CONDition? (see page 1015)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:USER:ENABle <sum_of_bits> (see page 1016)	:STATus:USER:ENABle? (see page 1016)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
:STATus:USER:NTRansition <sum_of_bits> (see page 1017)	:STATus:USER:NTRansition? (see page 1017)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format

Table 130 :STATus:USER Commands Summary (continued)

Command	Query	Options and Query Returns
:STATus:USER:PTRansition <sum_of_bits> (see page 1018)	:STATus:USER:PTRansition? (see page 1018)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format
n/a	:STATus:USER[:EVENt]? (see page 1019)	<sum_of_bits> ::= 0-32767; an integer in NR1 format <b> ::= 0-14; an integer in NR1 format

User Event Register

To see where the User Event Register appears in the status register hierarchy, see Chapter 41, “Status Reporting,” starting on page 1253.

To unserstand the SCPI status register model, see "Introduction to :STATus Commands" on page 883.

The following bits appear in the User Event Register.

Table 131 User Event Register Bits

Bit	Name	Description	When Set (1 = High = True), Indicates:
15	---	---	(Not used, always zero.)
14	---	---	(Not used.)
13	---	---	(Not used.)
12	---	---	(Not used.)
11	---	---	(Not used.)
10	---	---	(Not used.)
9	---	---	(Not used.)
8	---	---	(Not used.)
7	---	---	(Not used.)
6	---	---	(Not used.)
5	---	---	(Not used.)
4	---	---	(Not used.)
3	---	---	(Not used.)
2	---	---	(Not used.)
1	---	---	(Not used.)
0	USR	User Event	An enabled user event condition has occurred.

When enabled, the summary message for the Trigger Event Register goes to Status Byte Register bit 1 (USR). See "*CLS (Clear Status)" on page 184.

:STATus:USER:BIT<b>:CONDition?

(see page 1292)

Query Syntax

```
:STATus:USER:BIT<b>:CONDition?

<b> ::= 0-14; an integer in NR1 format
```

The :STATus:USER:BIT<b>:CONDition? query returns the value of a particular bit in the Condition register.

The Condition register continuously monitors status. Reading a Condition register bit does not affect its contents.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:USER:BIT<b>:ENABle" on page 1011
- ":STATus:USER:BIT<b>:NTRansition" on page 1012
- ":STATus:USER:BIT<b>:PTRansition" on page 1013
- ":STATus:USER:BIT<b>[:EVENt]?" on page 1014
- ":STATus:USER:CONDition?" on page 1015
- ":STATus:USER:ENABle" on page 1016
- ":STATus:USER:NTRansition" on page 1017
- ":STATus:USER:PTRansition" on page 1018
- ":STATus:USER[:EVENt]?" on page 1019
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:USER:BIT<b>:ENABle

 (see page 1292)

Command Syntax

```
:STATus:USER:BIT<b>:ENABle <bit_value>

<b> ::= 0-14; an integer in NR1 format

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

The :STATus:USER:BIT<b>:ENABle command sets a particular bit in the Enable register that identifies a corresponding Event register bit that will be ORed with other identified Event register bits to create the Summary Message bit that is sent to the parent status register.

Query Syntax

```
:STATus:USER:BIT<b>:ENABle?
```

The :STATus:USER:BIT<b>:ENABle? query returns the value of the specified Enable register bit.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:USER:BIT<b>:CONDition?" on page 1010
- ":STATus:USER:BIT<b>:NTRansition" on page 1012
- ":STATus:USER:BIT<b>:PTRansition" on page 1013
- ":STATus:USER:BIT<b>[:EVENt]?" on page 1014
- ":STATus:USER:CONDition?" on page 1015
- ":STATus:USER:ENABle" on page 1016
- ":STATus:USER:NTRansition" on page 1017
- ":STATus:USER:PTRansition" on page 1018
- ":STATus:USER[:EVENt]?" on page 1019
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:USER:BIT<b>:NTRansition

N (see page 1292)

Command Syntax

```
:STATus:USER:BIT<b>:NTRansition <bit_value>

<b> ::= 0-14; an integer in NR1 format

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

The :STATus:USER:BIT<b>:NTRansition command sets a particular bit in the Negative Transition filter that identifies a Condition register bit on which a transition from 1 to 0 or TRUE to FALSE will be latched into the corresponding Event register bit.

Query Syntax

```
:STATus:USER:BIT<b>:NTRansition?
```

The :STATus:USER:BIT<b>:NTRansition? query returns the value of the specified Negative Transition filter bit.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:USER:BIT<b>:CONDition?" on page 1010
- ":STATus:USER:BIT<b>:ENABle" on page 1011
- ":STATus:USER:BIT<b>:PTRansition" on page 1013
- ":STATus:USER:BIT<b>[:EVENt]?" on page 1014
- ":STATus:USER:CONDition?" on page 1015
- ":STATus:USER:ENABle" on page 1016
- ":STATus:USER:NTRansition" on page 1017
- ":STATus:USER:PTRansition" on page 1018
- ":STATus:USER[:EVENt]?" on page 1019
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:USER:BIT<b>:PTRansition

(see page 1292)

Command Syntax

```
:STATus:USER:BIT<b>:PTRansition <bit_value>

<b> ::= 0-14; an integer in NR1 format

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

The :STATus:USER:BIT<b>:PTRansition command sets a particular bit in the Positive Transition filter that identifies a Condition register bit on which a transition from 0 to 1 or FALSE to TRUE will be latched into the corresponding Event register bit.

Query Syntax

```
:STATus:USER:BIT<b>:PTRansition?
```

The :STATus:USER:BIT<b>:PTRansition? query returns the value of the specified Positive Transition filter bit.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:USER:BIT<b>:CONDition?" on page 1010
- ":STATus:USER:BIT<b>:ENABle" on page 1011
- ":STATus:USER:BIT<b>:NTRansition" on page 1012
- ":STATus:USER:BIT<b>[:EVENt]?" on page 1014
- ":STATus:USER:CONDition?" on page 1015
- ":STATus:USER:ENABle" on page 1016
- ":STATus:USER:NTRansition" on page 1017
- ":STATus:USER:PTRansition" on page 1018
- ":STATus:USER[:EVENt]?" on page 1019
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:USER:BIT<b>[:EVENt]?

(see page 1292)

Query Syntax

```
:STATus:USER:BIT<b>[:EVENt]?

<b> ::= 0-14; an integer in NR1 format
```

The :STATus:USER:BIT<b>[:EVENt]? query returns the value of the specified Event register bit and clears the bit.

Return Format

```
<bit_value><NL>

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

See Also

- ":STATus:USER:BIT<b>:CONDition?" on page 1010
- ":STATus:USER:BIT<b>:ENABle" on page 1011
- ":STATus:USER:BIT<b>:NTRansition" on page 1012
- ":STATus:USER:BIT<b>:PTRansition" on page 1013
- ":STATus:USER:CONDition?" on page 1015
- ":STATus:USER:ENABle" on page 1016
- ":STATus:USER:NTRansition" on page 1017
- ":STATus:USER:PTRansition" on page 1018
- ":STATus:USER[:EVENt]?" on page 1019
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:USER:CONDition?

(see page 1292)

Query Syntax `:STATus:USER:CONDition?`

The :STATus:USER:CONDition? query returns the decimal value of the sum of the bits in the Condition register.

The Condition register continuously monitors status. Reading the Condition register does not affect its contents.

Return Format `<sum_of_bits><NL>`

`<sum_of_bits> ::= 0-32767; an integer in NR1 format`

See Also

- ":STATus:USER:BIT<b>:CONDition?" on page 1010
- ":STATus:USER:BIT<b>:ENABle" on page 1011
- ":STATus:USER:BIT<b>:NTRansition" on page 1012
- ":STATus:USER:BIT<b>:PTRansition" on page 1013
- ":STATus:USER:BIT<b>[:EVENt]?" on page 1014
- ":STATus:USER:ENABle" on page 1016
- ":STATus:USER:NTRansition" on page 1017
- ":STATus:USER:PTRansition" on page 1018
- ":STATus:USER[:EVENt]?" on page 1019
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:USER:ENABle

(see page 1292)

Command Syntax

```
:STATus:USER:ENABle <sum_of_bits>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

The :STATus:USER:ENABle command sets bits in the Enable register that identify which Event register bits are ORed to create the Summary Message bit that is sent to the parent status register.

Query Syntax

```
:STATus:USER:ENABle?
```

The :STATus:USER:ENABle? query returns the decimal value of the sum of the bits in the Enable register.

Return Format

```
<sum_of_bits><NL>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

See Also

- ":STATus:USER:BIT<b>:CONDition?" on page 1010
- ":STATus:USER:BIT<b>:ENABle" on page 1011
- ":STATus:USER:BIT<b>:NTRansition" on page 1012
- ":STATus:USER:BIT<b>:PTRansition" on page 1013
- ":STATus:USER:BIT<b>[:EVENt]?" on page 1014
- ":STATus:USER:CONDition?" on page 1015
- ":STATus:USER:NTRansition" on page 1017
- ":STATus:USER:PTRansition" on page 1018
- ":STATus:USER[:EVENt]?" on page 1019
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:USER:NTRansition

N (see page 1292)

Command Syntax

```
:STATus:USER:NTRansition <sum_of_bits>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

The :STATus:USER:NTRansition command sets bits in the Negative Transition filter that identify the Condition register bits on which transitions from 1 to 0 or TRUE to FALSE will be latched into the Event register.

Query Syntax

```
:STATus:USER:NTRansition?
```

The :STATus:USER:NTRansition? query returns the decimal value of the sum of the bits in the Negative Transition filter.

Return Format

```
<sum_of_bits><NL>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

See Also

- ":STATus:USER:BIT<b>:CONDition?" on page 1010
- ":STATus:USER:BIT<b>:ENABle" on page 1011
- ":STATus:USER:BIT<b>:NTRansition" on page 1012
- ":STATus:USER:BIT<b>:PTRansition" on page 1013
- ":STATus:USER:BIT<b>[:EVENt]?" on page 1014
- ":STATus:USER:CONDition?" on page 1015
- ":STATus:USER:ENABle" on page 1016
- ":STATus:USER:PTRansition" on page 1018
- ":STATus:USER[:EVENt]?" on page 1019
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:USER:PTRansition

(see page 1292)

Command Syntax

```
:STATus:USER:PTRansition <sum_of_bits>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

The :STATus:USER:PTRansition command sets bits in the Positive Transition filter that identify the Condition register bits on which transitions from 0 to 1 or FALSE to TRUE will be latched into the Event register.

Query Syntax

```
:STATus:USER:PTRansition?
```

The :STATus:USER:PTRansition? query returns the decimal value of the sum of the bits in the Positive Transition filter.

Return Format

```
<sum_of_bits><NL>

<sum_of_bits> ::= 0-32767; an integer in NR1 format
```

See Also

- ":STATus:USER:BIT<b>:CONDition?" on page 1010
- ":STATus:USER:BIT<b>:ENABle" on page 1011
- ":STATus:USER:BIT<b>:NTRansition" on page 1012
- ":STATus:USER:BIT<b>:PTRansition" on page 1013
- ":STATus:USER:BIT<b>[:EVENt]?" on page 1014
- ":STATus:USER:CONDition?" on page 1015
- ":STATus:USER:ENABle" on page 1016
- ":STATus:USER:NTRansition" on page 1017
- ":STATus:USER[:EVENt]?" on page 1019
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

:STATus:USER[:EVENt]?

N (see page 1292)

Query Syntax `:STATus:USER[:EVENt]?`

The :STATus:USER[:EVENt]? query returns the decimal value of the sum of the bits in the Event register and clears the register.

The contents of the Event register show the events that have occurred since the last time the register was read.

Return Format `<sum_of_bits><NL>`

`<sum_of_bits> ::= 0-32767; an integer in NR1 format`

See Also

- ":STATus:USER:BIT<b>:CONDition?" on page 1010
- ":STATus:USER:BIT<b>:ENABle" on page 1011
- ":STATus:USER:BIT<b>:NTRansition" on page 1012
- ":STATus:USER:BIT<b>:PTRansition" on page 1013
- ":STATus:USER:BIT<b>[:EVENt]?" on page 1014
- ":STATus:USER:CONDition?" on page 1015
- ":STATus:USER:ENABle" on page 1016
- ":STATus:USER:NTRansition" on page 1017
- ":STATus:USER:PTRansition" on page 1018
- ":STATus:PRESet" on page 887
- "*CLS (Clear Status)" on page 184

33 :SYSTem Commands

Control basic system functions of the oscilloscope. See "Introduction to :SYSTem Commands" on page 1024.

Table 132 :SYSTem Commands Summary

Command	Query	Options and Query Returns
:SYSTem:DATE <date> (see page 1025)	:SYSTem:DATE? (see page 1025)	<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}
n/a	:SYSTem:DIDentifier? (see page 1026)	n/a
:SYSTem:DSP <string> (see page 1027)	n/a	<string> ::= up to 75 characters as a quoted ASCII string
n/a	:SYSTem:ERRor:ALL? (see page 1028)	<errors_list> ::= comma-separated list of all errors from the queue (codes and strings)
n/a	:SYSTem:ERRor:CODE[:NEXT]? (see page 1029)	<error_code> ::= the next error (code only) from the queue
n/a	:SYSTem:ERRor:COUNt? (see page 1030)	<count> ::= an integer number of errors in the queue
n/a	:SYSTem:ERRor:EXTended? <type> (see page 1031)	<type> ::= {INFO | WARNING | ERROR | FATAL | MESSAGE | ALL} <errors_list> ::= returns a comma-separated list of all errors (strings, no codes) of the specified type from the queue

Table 132 :SYSTem Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:SYSTem:ERRor[:NEXT]? (see page 1032)	<error_number>,<error_string> ::= the next error (code and string) from the queue <error_number> ::= an integer error code <error string> ::= quoted ASCII string. See Error Messages (see page 1245).
n/a	:SYSTem:HELP:HEADers? (see page 1034)	<binary_block> ::= newline-separated list of supported SCPI commands/queries
n/a	:SYSTem:HID? (see page 1035)	n/a
:SYSTem:LOCK <value> (see page 1036)	:SYSTem:LOCK? (see page 1036)	<value> ::= {{1 | ON} | {0 | OFF}}
n/a	:SYSTem:LOCK:OWNer? (see page 1037)	<session_string> ::= quoted session string or "NONE"
:SYSTem:LOCK:RELease (see page 1038)	n/a	n/a
n/a	:SYSTem:LOCK:REQuest? (see page 1039)	{1 | 0} (for granted or not)
:SYSTem:PERSona[:MANufacturer] <manufacturer_string> (see page 1040)	:SYSTem:PERSona[:MANufacturer]? (see page 1040)	<manufacturer_string> ::= quoted ASCII string, up to 63 characters
:SYSTem:PERSona[:MANufacturer]:DEFault (see page 1041)	n/a	Sets manufacturer string to "KEYSIGHT TECHNOLOGIES"
:SYSTem:POWercycle (see page 1042)	n/a	n/a
:SYSTem:PRECision:LENGth <length> (see page 1043)	:SYSTem:PRECision:LENGth? (see page 1043)	<length> ::= {64K | 128K | 256K | 512K | 1M | 1.5M | 2M | 4M | 8M | 12M | 16M | 24M | 32M}
:SYSTem:PRESet (see page 1044)	n/a	See :SYSTem:PRESet (see page 1044)
:SYSTem:PROTection:LOCK <value> (see page 1047)	:SYSTem:PROTection:LOCK? (see page 1047)	<value> ::= {{1 | ON} | {0 | OFF}}

Table 132 :SYSTem Commands Summary (continued)

Command	Query	Options and Query Returns
:SYSTem:RLOGger <setting>[,<file_name>[,<write_mode>]] (see page 1048)	n/a	<setting> ::= {{0 | OFF} | {1 | ON}} <file_name> ::= quoted ASCII string <write_mode> ::= {CREate | APPend}
:SYSTem:RLOGger:DESTination <dest> (see page 1049)	:SYSTem:RLOGger:DESTination? (see page 1049)	<dest> ::= {FILE | SCReen | BOTH}
:SYSTem:RLOGger:DISPlay {{0 | OFF} | {1 | ON}} (see page 1050)	:SYSTem:RLOGger:DISPlay? (see page 1050)	<setting> ::= {0 | 1}
:SYSTem:RLOGger:FNAMe <file_name> (see page 1051)	:SYSTem:RLOGger:FNAMe? (see page 1051)	<file_name> ::= quoted ASCII string
:SYSTem:RLOGger:STATe {{0 | OFF} | {1 | ON}} (see page 1052)	:SYSTem:RLOGger:STATe? (see page 1052)	<setting> ::= {0 | 1}
:SYSTem:RLOGger:TRANsparent {{0 | OFF} | {1 | ON}} (see page 1053)	:SYSTem:RLOGger:TRANsparent? (see page 1053)	<setting> ::= {0 | 1}
:SYSTem:RLOGger:WMODe <write_mode> (see page 1054)	:SYSTem:RLOGger:WMODe? (see page 1054)	<write_mode> ::= {CREate | APPend}
:SYSTem:SETup <setup_data> (see page 1055)	:SYSTem:SETup? (see page 1055)	<setup_data> ::= data in IEEE 488.2 # format.
:SYSTem:SHUTdown (see page 1057)	n/a	n/a
:SYSTem:TIME <time> (see page 1058)	:SYSTem:TIME? (see page 1058)	<time> ::= hours,minutes,seconds in NR1 format
:SYSTem:TIME:DSAVing <value> (see page 1059)	:SYSTem:TIME:DSAVing? (see page 1059)	<value> ::= {{1 | ON} | {0 | OFF}}
:SYSTem:TIME:NTPRotocol {{0 | OFF} | {1 | ON}} (see page 1060)	:SYSTem:TIME:NTPRotocol? (see page 1060)	<setting> ::= {0 | 1}

Table 132 :SYSTem Commands Summary (continued)

Command	Query	Options and Query Returns
:SYSTem:TIME:ZONE <UTC_P_or_M_HHMM_offset> (see page 1061)	:SYSTem:TIME:ZONE? (see page 1061)	<UTC_P_or_M_HHMM_offset> ::= {UP0000 | UP0100 | UP0200 | UP0300 | UP0330 | UP0400 | UP0430 | UP0500 | UP0530 | UP0600 | UP0700 | UP0800 | UP0845 | UP0900 | UP0930 | UP1000 | UP1030 | UP1100 | UP1200 | UM0100 | UM0200 | UM0300 | UM0330 | UM0400 | UM0500 | UM0600 | UM0700 | UM0800 | UM0900 | UM1000 | UM1100 | UM1200}
:SYSTem:TOUCh {{1 | ON} | {0 | OFF}} (see page 1062)	:SYSTem:TOUCh? (see page 1062)	{1 | 0}
n/a	:SYSTem:VERSion? (see page 1063)	<SCPI_version_implemented> ::= 1999.0

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:DATE

N (see page 1292)

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" on page 1024
- ":SYSTem:TIME" on page 1058

:SYSTem:DIDentifier?

(see page 1292)

Query Syntax `:SYSTem:DIDentifier?`

The :SYSTem:DIDentifier? query returns the oscilloscope's Host ID string.

The oscilloscope's Host ID is needed when redeeming licenses for oscilloscope upgrades or other licensed features.

The exact format of returned string are product-specific. This example returns a model number and serial number in a comma-separated format:

```
X12345A,US12345678
```

Portable programs should not attempt to parse the contents of the returned string. Use the *IDN? query instead to get the model number and/or serial number in a defined portable format.

NOTE This query is the same as the :SYSTem:HID? query.

Return Format

```
<host_id><NL>

<host_id> ::= ASCII string
```

See Also

- "*IDN? (Identification Number)" on page 189
- ":SYSTem:HID?" on page 1035
- "Queries Not Supported in Multiple Program Message Units" on page 1297

:SYSTem:DSP

N (see page 1292)

Command Syntax

```
:SYSTem:DSP <string>

<string> ::= quoted ASCII string (up to 75 characters)
```

The :SYSTem:DSP command writes the quoted string (excluding quotation marks) to a text box on-screen.

Use :SYSTem:DSP "" to remotely remove the message from the display and save it in the message log. (Two sets of quote marks without a space between them creates a NULL string.)

Also, to manually remove the message from the display and save it in the message log, you can:

- Press any front panel key.
- Select the close button ("X") in the graphical user interface's message dialog box.

See Also

- "Introduction to :SYSTem Commands" on page 1024

:SYSTem:ERRor:ALL?

(see page 1292)

Query Syntax `:SYSTem:ERRor:ALL?`

The :SYSTem:ERRor:ALL? query returns a comma-separated list of all errors from the queue (codes and strings).

Return Format `<errors_list><NL>`

```
<errors_list> ::= comma-separated list of all errors from the queue
   (codes and strings)
```

See Also

- ":SYSTem:ERRor:CODE[:NEXT]?" on page 1029
- ":SYSTem:ERRor:COUNt?" on page 1030
- ":SYSTem:ERRor:EXTended?" on page 1031
- ":SYSTem:ERRor[:NEXT]?" on page 1032

:SYSTem:ERRor:CODE[:NEXT]?

(see page 1292)

Query Syntax `:SYSTem:ERRor:CODE[:NEXT]?`

The :SYSTem:ERRor:CODE[:NEXT]? query returns the next error (code only) from the queue.

Except for the shortened response, the query operates identically to the SYSTem:ERRor[:NEXT]? query.

Return Format `<error_code><NL>`

`<error_code> ::= the next error (code only) from the queue`

See Also

- ":SYSTem:ERRor:ALL?" on page 1028
- ":SYSTem:ERRor:COUNt?" on page 1030
- ":SYSTem:ERRor:EXTended?" on page 1031
- ":SYSTem:ERRor[:NEXT]?" on page 1032

:SYSTem:ERRor:COUNt?

(see page 1292)

Query Syntax `:SYSTem:ERRor:COUNt?`

The :SYSTem:ERRor:COUNt? query returns an integer number of errors in the queue.

Return Format `<count><NL>`

`<count> ::= an integer number of errors in the queue`

See Also

- ":SYSTem:ERRor:ALL?" on page 1028
- ":SYSTem:ERRor:CODE[:NEXT]?" on page 1029
- ":SYSTem:ERRor:EXTended?" on page 1031
- ":SYSTem:ERRor[:NEXT]?" on page 1032

:SYSTem:ERRor:EXTended?

(see page 1292)

Query Syntax

```
:SYSTem:ERRor:EXTended? <type>

<type> ::= {INFO | WARNING | ERROR | FATAL | MESSAGE | ALL}
```

The :SYSTem:ERRor:EXTended? query returns a comma-separated list of all errors (strings, no codes) of the specified type from the queue.

Return Format

```
<errors_list><NL>

<errors_list> ::= comma-separated list of all errors (strings,
   no codes) of the specified type from the queue
```

See Also

- ":SYSTem:ERRor:ALL?" on page 1028
- ":SYSTem:ERRor:CODE[:NEXT]?" on page 1029
- ":SYSTem:ERRor:COUNt?" on page 1030
- ":SYSTem:ERRor[:NEXT]?" on page 1032

:SYSTem:ERRor[:NEXT]?

(see page 1292)

Query Syntax

```
:SYSTem:ERRor[:NEXT]?
```

The :SYSTem:ERRor[:NEXT]? 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[:NEXT]? 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.

When remote logging is enabled (using the oscilloscope's front panel), additional debug information can be included in the returned error string. If the error is detected by the SCPI command parser, such as a header error or other syntax error, the extra debug information is generated and included. But if the error is detected by the oscilloscope system, such as when an out-of-range value is sent, then no extra debug information is included.

Error Queue

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).

When the error queue is read with the :SYSTem:ERRor[:NEXT]? query, the oldest error is removed 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.

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 Chapter 40, “Error Messages,” starting on page 1245.

See Also

- "Introduction to :SYSTem Commands" on page 1024
- "*ESR? (Standard Event Status Register)" on page 187
- "*CLS (Clear Status)" on page 184
- ":SYSTem:ERRor:ALL?" on page 1028
- ":SYSTem:ERRor:CODE[:NEXT]?" on page 1029

- ":SYSTem:ERRor:COUNt?" on page 1030
- ":SYSTem:ERRor:EXTended?" on page 1031

:SYSTem:HELP:HEADers?

N (see page 1292)

Query Syntax `:SYSTem:HELP:HEADers?`

The :SYSTem:HELP:HEADers? query returns a binary block that contains a newline-separated list of supported SCPI commands and queries.

Return Format `<binary_block><NL>`

`<binary_block> ::= newline-separated list of supported SCPI commands/queries`

Lines can have the following suffixes:

- `/nquery/` — There is no "query" version of the SCPI header.
- `/qonly/` — There is no "command" version of the SCPI header.

For example, a portion of the returned result looks like:

```
...
*SRE
*STB?/qonly/
*TRG/nquery/
*TST?/qonly/
*WAI/nquery/
:ABORt/nquery/
:ACQuire:COMPlete
:ACQuire:COUNt
:ACQuire:DIGitizer
:ACQuire:MANual
:ACQuire:MODE
:ACQuire:POINts[:ANALog]
:ACQuire:POINts[:ANALog]:AUTO
:ACQuire:RSIGnal
...
```

Square brackets in a SCPI header show optional parts of a command/query. The previous example shows `:ACQuire:POINts:AUTO` is the same as `:ACQuire:POINts:ANALog:AUTO`.

:SYSTem:HID?

(see page 1292)

Query Syntax `:SYSTem:HID?`

The :SYSTem:HID? query returns the oscilloscope's Host ID string.

The oscilloscope's Host ID is needed when redeeming licenses for oscilloscope upgrades or other licensed features.

The exact format of returned string are product-specific. This example returns a model number and serial number in a comma-separated format:

```
X12345A,US12345678
```

Portable programs should not attempt to parse the contents of the returned string. Use the *IDN? query instead to get the model number and/or serial number in a defined portable format.

NOTE This query is the same as the :SYSTem:DIDentifier? query.

Return Format

```
<host_id><NL>

<host_id> ::= ASCII string
```

See Also

- "*IDN? (Identification Number)" on page 189
- ":SYSTem:DIDentifier?" on page 1026
- "Queries Not Supported in Multiple Program Message Units" on page 1297

:SYSTem:LOCK

N (see page 1292)

Command Syntax

```
:SYSTem:LOCK <on_off>

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

The :SYSTem:LOCK command disables the front panel (and touch screen). LOCK ON is the equivalent of sending a local lockout message over the programming interface.

Query Syntax

```
:SYSTem:LOCK?
```

The :SYSTem:LOCK? query returns the lock status of the front panel.

Return Format

```
<on_off><NL>

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

See Also

- "Introduction to :SYSTem Commands" on page 1024

:SYSTem:LOCK:OWNer?

(see page 1292)

Query Syntax `:SYSTem:LOCK:OWNer?`

The :SYSTem:LOCK:OWNer? query returns what session currently has the lock on this device. If no session has a lock then "NONE" is returned.

Return Format `<session_string><NL>`

`<session_string> ::= quoted session string or "NONE"`

See Also

- ":SYSTem:LOCK:RELease" on page 1038
- ":SYSTem:LOCK:REQuest?" on page 1039

:SYSTem:LOCK:RELease

(see page 1292)

Command Syntax `:SYSTem:LOCK:RELease`

The :SYSTem:LOCK:RELease command releases the lock if owned by this session.

If this session does not have the lock, this command has no effect.

See Also

- ":SYSTem:LOCK:OWNer?" on page 1037
- ":SYSTem:LOCK:REQuest?" on page 1039

:SYSTem:LOCK:REQuest?

(see page 1292)

Query Syntax `:SYSTem:LOCK:REQuest?`

The :SYSTem:LOCK:REQuest? query attempts to attain the lock on this device and returns 1 if successful or 0 if it fails.

Return Format `<grant_status><NL>`

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

See Also

- ":SYSTem:LOCK:OWNer?" on page 1037
- ":SYSTem:LOCK:RELease" on page 1038

:SYSTem:PERSona[:MANufacturer]

 (see page 1292)

Command Syntax

```
:SYSTem:PERSona[:MANufacturer] <manufacturer_string>

<manufacturer_string> ::= ::= quoted ASCII string, up to 63 characters
```

The :SYSTem:PERSona[:MANufacturer] command lets you change the manufacturer string portion of the identification string returned by the *IDN? query.

The default manufacturer string is "KEYSIGHT TECHNOLOGIES".

If your remote programs depend on a legacy manufacturer string, for example, you could use this command to set the manufacturer string to "AGILENT TECHNOLOGIES".

Query Syntax

```
:SYSTem:PERSona[:MANufacturer]?
```

The :SYSTem:PERSona[:MANufacturer]? query returns the currently set manufacturer string.

Return Format

```
<manufacturer_string><NL>
```

See Also

- "*IDN? (Identification Number)" on page 189
- ":SYSTem:PERSona[:MANufacturer]:DEFault" on page 1041
- "Introduction to :SYSTem Commands" on page 1024

:SYSTem:PERSona[:MANufacturer]:DEFault

 (see page 1292)

Command Syntax `:SYSTem:PERSona[:MANufacturer]:DEFault`

The :SYSTem:PERSona[:MANufacturer]:DEFault command sets the manufacturer string to "KEYSIGHT TECHNOLOGIES".

See Also

- "*IDN? (Identification Number)" on page 189
- ":SYSTem:PERSona[:MANufacturer]" on page 1040
- "Introduction to :SYSTem Commands" on page 1024

:SYSTem:POWercycle

 (see page 1292)

Command Syntax `:SYSTem:POWercycle`

The :SYSTem:POWercycle command powers down the oscilloscope and restarts it. This is the same as pressing the front panel power button to power down the oscilloscope and then pressing the power button again to start the oscilloscope.

See Also

- ":SYSTem:SHUTdown" on page 1057

:SYSTem:PRECision:LENGth

(see page 1292)

Command Syntax

```
:SYSTem:PRECision:LENGth <length>

<length> ::= {64K | 128K | 256K | 512K | 1M | 1.5M | 2M | 4M | 8M
   | 12M | 16M | 24M | 32M}
```

The :SYSTem:PRECision:LENGth command specifies the length of the analysis record. You can specify 64K, 128K, 256K, 512K, 1M, 1.5M, 2M, 4M, 8M, 12M, 16M, 24M, or 32M points unless limited by the oscilloscope's overall memory depth.

Query Syntax

```
:SYSTem:PRECision:LENGth?
```

The :SYSTem:PRECision:LENGth? query returns the current analysis record length setting.

Return Format

```
<length><NL>

<length> ::= {64K | 128K | 256K | 512K | 1M | 1.5M | 2M | 4M | 8M
   | 12M | 16M | 24M | 32M}
```

See Also

- "Introduction to :SYSTem Commands" on page 1024
- ":WAVeform:POINts:MODE" on page 1158
- "*RST (Reset)" on page 194

:SYSTem:PRESet

C (see page 1292)

Command Syntax `:SYSTem:PRESet`

The :SYSTem:PRESet command places the instrument in a known state. This is the same as pressing the **[Default Setup]** key or selecting **Control > Default Setup** in the graphical user interface.

When you perform a default setup, some user settings (like preferences) remain unchanged. To reset all user settings to their factory defaults, use the *RST command.

Reset conditions are:

Acquire Settings	
Mode	Normal
Averaging	Off
# Averages	8

Analog Channel Settings	
Channel 1	On
Channel 2/3/4	Off
Volts/division	5.00 V
Offset	0.00
Coupling	DC
Probe attenuation	10:1
Vernier	Off
Invert	Off
BW limit	Off
Impedance	1 M Ohm (cannot be changed)
Units	Volts
Skew	0

Cursor Settings	
Source	Channel 1

Digital Channel Settings	
Channel 0 - 7	Off
Labels	Off
Threshold	TTL (1.4 V)

Display Settings	
Persistence	Off
Grid	20%

Measurement Settings	
Source	Channel 1

Run Control Settings	
	Scope is running

Time Base Settings	
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 Settings	
Type	Edge
Mode	Auto
Coupling	dc
Source	Channel 1
Level	0.0 V
Slope	Positive

Trigger Settings	
HF Reject and noise reject	Off
Holdoff	40 ns
External probe attenuation	1:1
External Units	Volts
External Impedance	1 M Ohm (cannot be changed)

See Also
- "Introduction to Common (*) Commands" on page 182
- "*RST (Reset)" on page 194

:SYSTem:PROTection:LOCK

N (see page 1292)

Command Syntax

```
:SYSTem:PROTection:LOCK <on_off>

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

The :SYSTem:PROTection:LOCK command disables the fifty ohm impedance setting for all analog channels.

NOTE

Be careful when turning ON the :SYSTem:PROTection:LOCK because there is no visual indication of this setting in the front-panel user interface (other than a disabled 50 Ω channel input impedance selection), and a user could think the oscilloscope is not working properly.

Query Syntax

```
:SYSTem:PROTection:LOCK?
```

The :SYSTem:PROTection:LOCK? query returns the analog channel protection lock status.

Return Format

```
<on_off><NL>

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

See Also

- "Introduction to :SYSTem Commands" on page 1024

:SYSTem:RLOGger

(see page 1292)

Command Syntax

```
:SYSTem:RLOGger <setting>[,<file_name>[,<write_mode>]]

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

<file_name> ::= quoted ASCII string

<write_mode> ::= {CREate | APPend}
```

The :SYSTem:RLOGger command enables or disables remote command logging, optionally specifying the log file name and write mode.

See Also

- ":SYSTem:RLOGger:DESTination" on page 1049
- ":SYSTem:RLOGger:DISPlay" on page 1050
- ":SYSTem:RLOGger:FNAMe" on page 1051
- ":SYSTem:RLOGger:STATe" on page 1052
- ":SYSTem:RLOGger:TRANsparent" on page 1053
- ":SYSTem:RLOGger:WMODe" on page 1054

:SYSTem:RLOGger:DESTination

(see page 1292)

Command Syntax

```
:SYSTem:RLOGger:DESTination <dest>

<dest> ::= {FILE | SCReen | BOTH}
```

The :SYSTem:RLOGger:DESTination command specifies whether remote commands are logged to a text file (on a connected USB storage device), logged to the screen, or both.

NOTE **If the destination is changed while remote command logging is running, remote command logging is turned off.**

Query Syntax

```
:SYSTem:RLOGger:DESTination?
```

The :SYSTem:RLOGger:DESTination? query returns the remote command logging destination.

Return Format

```
<dest><NL>

<dest> ::= {FILE | SCR | BOTH}
```

See Also

- ":SYSTem:RLOGger" on page 1048
- ":SYSTem:RLOGger:DISPlay" on page 1050
- ":SYSTem:RLOGger:FNAMe" on page 1051
- ":SYSTem:RLOGger:STATe" on page 1052
- ":SYSTem:RLOGger:TRANsparent" on page 1053
- ":SYSTem:RLOGger:WMODe" on page 1054

:SYSTem:RLOGger:DISPlay

 (see page 1292)

Command Syntax `:SYSTem:RLOGger:DISPlay {{0 | OFF} | {1 | ON}}`

The :SYSTem:RLOGger:DISPlay command enables or disables the screen display of logged remote commands and their return values (if applicable).

Query Syntax `:SYSTem:RLOGger:DISPlay?`

The :SYSTem:RLOGger:DISPlay? query returns whether the screen display for remote command logging is enabled or disabled.

Return Format `<setting><NL>`

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

See Also

- ":SYSTem:RLOGger" on page 1048
- ":SYSTem:RLOGger:DESTination" on page 1049
- ":SYSTem:RLOGger:FNAMe" on page 1051
- ":SYSTem:RLOGger:STATe" on page 1052
- ":SYSTem:RLOGger:TRANsparent" on page 1053
- ":SYSTem:RLOGger:WMODe" on page 1054

:SYSTem:RLOGger:FNAMe

(see page 1292)

Command Syntax

```
:SYSTem:RLOGger:FNAMe <file_name>

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

The :SYSTem:RLOGger:FNAMe command specifies the remote command log file name.

Because log files are ASCII text files, the ".txt" extension is automatically added to the name specified.

Query Syntax

```
:SYSTem:RLOGger:FNAMe?
```

The :SYSTem:RLOGger:FNAMe? query returns the remote command log file name.

Return Format

```
<file_name><NL>
```

See Also

- ":SYSTem:RLOGger" on page 1048
- ":SYSTem:RLOGger:DESTination" on page 1049
- ":SYSTem:RLOGger:DISPlay" on page 1050
- ":SYSTem:RLOGger:STATe" on page 1052
- ":SYSTem:RLOGger:TRANsparent" on page 1053
- ":SYSTem:RLOGger:WMODe" on page 1054

:SYSTem:RLOGger:STATe

(see page 1292)

Command Syntax `:SYSTem:RLOGger:STATe {{0 | OFF} | {1 | ON}}`

The :SYSTem:RLOGger:STATe command enables or disables remote command logging.

Query Syntax `:SYSTem:RLOGger:STATe?`

The :SYSTem:RLOGger:STATe? query returns the remote command logging state.

Return Format `<setting><NL>`

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

See Also

- ":SYSTem:RLOGger" on page 1048
- ":SYSTem:RLOGger:DESTination" on page 1049
- ":SYSTem:RLOGger:DISPlay" on page 1050
- ":SYSTem:RLOGger:FNAMe" on page 1051
- ":SYSTem:RLOGger:TRANsparent" on page 1053
- ":SYSTem:RLOGger:WMODe" on page 1054

:SYSTem:RLOGger:TRANsparent

N (see page 1292)

Command Syntax `:SYSTem:RLOGger:TRANsparent {{0 | OFF} | {1 | ON}}`

The :SYSTem:RLOGger:TRANsparent command specifies whether the screen display background for remote command logging is transparent or solid.

Query Syntax `:SYSTem:RLOGger:TRANsparent?`

The :SYSTem:RLOGger:TRANsparent? query returns the setting for transparent screen display background.

Return Format `<setting><NL>`

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

See Also

- ":SYSTem:RLOGger" on page 1048
- ":SYSTem:RLOGger:DESTination" on page 1049
- ":SYSTem:RLOGger:DISPlay" on page 1050
- ":SYSTem:RLOGger:FNAMe" on page 1051
- ":SYSTem:RLOGger:STATe" on page 1052
- ":SYSTem:RLOGger:WMODe" on page 1054

:SYSTem:RLOGger:WMODe

(see page 1292)

Command Syntax

```
:SYSTem:RLOGger:WMODe <write_mode>

<write_mode> ::= {CREate | APPend}
```

The :SYSTem:RLOGger:WMODe command specifies the remote command logging write mode.

Query Syntax

```
:SYSTem:RLOGger:WMODe?
```

The :SYSTem:RLOGger:WMODe? query returns the remote command logging write mode.

Return Format

```
<write_mode><NL>

<write_mode> ::= {CRE | APP}
```

See Also

- ":SYSTem:RLOGger" on page 1048
- ":SYSTem:RLOGger:DESTination" on page 1049
- ":SYSTem:RLOGger:DISPlay" on page 1050
- ":SYSTem:RLOGger:FNAMe" on page 1051
- ":SYSTem:RLOGger:STATe" on page 1052
- ":SYSTem:RLOGger:TRANsparent" on page 1053

:SYSTem:SETup

(see page 1292)

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 in IEEE 488.2 # format
```

See Also

- "Introduction to :SYSTem Commands" on page 1024
- "*LRN? (Learn Device Setup)" on page 190

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,
' #800075595<setup string><NL>
' where the setup string is 75595 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.scp"

' Open file for output.
Dim hFile As Integer
hFile = FreeFile
Open strPath For Binary Access Write Lock Write As hFile
Put hFile, , varQueryResult   ' Write data.
Close hFile   ' 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.scp"

' Open file for input.
Open strPath For Binary Access Read As hFile
Get hFile, , varSetupString   ' Read data.
Close hFile   ' Close file.
```

```
' Write setup string back to oscilloscope using ":SYSTem:SETup"
' command:
myScope.WriteIEEEBlock ":SYSTem:SETup ", varSetupString
CheckForInstrumentErrors
```

See complete example programs at: Chapter 44, “Programming Examples,” starting on page 1301

:SYSTem:SHUTdown

 (see page 1292)

Command Syntax `:SYSTem:SHUTdown`

The :SYSTem:SHUTdown command powers down the oscilloscope. This is the same as pressing the front panel power button to power down the oscilloscope. This allows the oscilloscope to shut down gracefully.

See Also

- ":SYSTem:POWercycle" on page 1042

:SYSTem:TIME

(see page 1292)

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" on page 1024
- ":SYSTem:DATE" on page 1025
- ":SYSTem:TIME:DSAVing" on page 1059
- ":SYSTem:TIME:NTPRotocol" on page 1060
- ":SYSTem:TIME:ZONE" on page 1061

:SYSTem:TIME:DSAVing

 (see page 1292)

Command Syntax

```
:SYSTem:TIME:DSAVing <on_off>

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

The :SYSTem:TIME:DSAVing command enables or disables the "automatically adjust clock for Daylight Savings Time" setting.

Query Syntax

```
:SYSTem:TIME:DSAVing?
```

The :SYSTem:TIME:DSAVing? query returns whether the "automatically adjust clock for Daylight Savings Time" setting is enabled or disabled.

Return Format

```
<on_off><NL>

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

See Also

- "Introduction to :SYSTem Commands" on page 1024
- ":SYSTem:TIME" on page 1058
- ":SYSTem:TIME:NTPRotocol" on page 1060
- ":SYSTem:TIME:ZONE" on page 1061

:SYSTem:TIME:NTPRotocol

(see page 1292)

Command Syntax `:SYSTem:TIME:NTPRotocol {{0 | OFF} | {1 | ON}}`

The :SYSTem:TIME:NTPRotocol command enables or disables automatically setting the oscilloscope's clock using Network Time Protocol.

Query Syntax `:SYSTem:TIME:NTPRotocol?`

The :SYSTem:TIME:NTPRotocol? query returns whether or not the oscilloscope's clock is being set using Network Time Protocol.

Return Format `<setting><NL>`

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

See Also

- ":SYSTem:TIME" on page 1058
- ":SYSTem:TIME:DSAVing" on page 1059
- ":SYSTem:TIME:ZONE" on page 1061

:SYSTem:TIME:ZONE

(see page 1292)

Command Syntax

```
:SYSTem:TIME:ZONE <UTC_P_or_M_HHMM_offset>

<UTC_P_or_M_HHMM_offset> ::= {UP0000 | UP0100 | UP0200 | UP0300
   | UP0330 | UP0400 | UP0430 | UP0500 | UP0530 | UP0600 | UP0700
   | UP0800 | UP0845 | UP0900 | UP0930 | UP1000 | UP1030 | UP1100
   | UP1200 | UM0100 | UM0200 | UM0300 | UM0330 | UM0400 | UM0500
   | UM0600 | UM0700 | UM0800 | UM0900 | UM1000 | UM1100 | UM1200}
```

The :SYSTem:TIME:ZONE command sets the system time zone based on the specified offset from UTC. This keeps local time constant and moves the UTC by setting the system clock to match the displayed oscilloscope clock.

Supported time zone offsets are: +0, +100, +200, +300, +330, +400, +430, +500, +530, +600, +700, +800, +845, +900, +930, +1000, +1030, +1100, +1200, -100, -200, -300, -330, -400, -500, -600, -700, -800, -900, -1000, -1100, -1200,

Because licenses are based on Coordinated Universal Time (UTC), the proper time zone setting allows licenses to work immediately after they are generated.

Query Syntax

```
:SYSTem:TIME:ZONE?
```

The :SYSTem:TIME:ZONE? query returns the time zone offset setting.

Return Format

```
<UTC_P_or_M_HHMM_offset><NL>
```

See Also

- ":SYSTem:TIME" on page 1058
- ":SYSTem:TIME:DSAVing" on page 1059
- ":SYSTem:TIME:NTPRotocol" on page 1060

:SYSTem:TOUCh

(see page 1292)

Command Syntax

```
:SYSTem:TOUCh <on_off>

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

The :SYSTem:TOUCh command disables or enables the touchscreen.

Query Syntax

```
:SYSTem:TOUCh?
```

The :SYSTem:TOUCh? query returns the touchscreen's on/off status.

Return Format

```
<on_off><NL>

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

See Also

- "Introduction to :SYSTem Commands" on page 1024

:SYSTem:VERSion?

(see page 1292)

Query Syntax `:SYSTem:VERSion?`

The :SYSTem:VERSion? query returns the version of the SCPI standard that is implemented by the oscilloscope.

Return Format `<SCPI_version_implemented><NL>`

`<SCPI_version_implemented> ::= 1999.0`

34 :TIMebase Commands

Control all horizontal sweep functions. See "Introduction to :TIMebase Commands" on page 1066.

Table 133 :TIMebase Commands Summary

Command	Query	Options and Query Returns
:TIMebase[:MAIN]:POSition <pos> (see page 1067)	:TIMebase[:MAIN]:POSition? (see page 1067)	<pos> ::= time from the trigger event to the display reference point in NR3 format
:TIMebase[:MAIN]:RANGe <range_value> (see page 1068)	:TIMebase[:MAIN]:RANGe? (see page 1068)	<range_value> ::= time for 10 div in seconds in NR3 format
:TIMebase[:MAIN]:SCALe <scale_value> (see page 1069)	:TIMebase[:MAIN]:SCALe? (see page 1069)	<scale_value> ::= time/div in seconds in NR3 format
:TIMebase:MODE <value> (see page 1070)	:TIMebase:MODE? (see page 1070)	<value> ::= {MAIN | WINDow}
:TIMebase:REFerence {LEFT | CENTer | RIGHt | CUSTom} (see page 1071)	:TIMebase:REFerence? (see page 1071)	<return_value> ::= {LEFT | CENTer | RIGHt | CUSTom}
:TIMebase:REFerence:LOCation <loc> (see page 1072)	:TIMebase:REFerence:LOCation? (see page 1072)	<loc> ::= 0.0 to 1.0 in NR3 format
:TIMebase:VERNier {{0 | OFF} | {1 | ON}} (see page 1073)	:TIMebase:VERNier? (see page 1073)	{0 | 1}
:TIMebase:WINDow:POSition <pos> (see page 1074)	:TIMebase:WINDow:POSition? (see page 1074)	<pos> ::= time from the trigger event to the zoomed view reference point in NR3 format

Table 133 :TIMebase Commands Summary (continued)

Command	Query	Options and Query Returns
:TIMebase:WINDow:RANGe <range_value> (see page 1075)	:TIMebase:WINDow:RANGe? (see page 1075)	<range value> ::= range value in seconds in NR3 format for the zoomed window
:TIMebase:WINDow:SCALe <scale_value> (see page 1076)	:TIMebase:WINDow:SCALe? (see page 1076)	<scale_value> ::= scale value in seconds in NR3 format for the zoomed 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 (zoomed) 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 +1.000E-03;POS +0.0E+00
```

:TIMebase[:MAIN]:POSition

(see page 1292)

Command Syntax

```
:TIMebase[:MAIN]:POSition <pos>

<pos> ::= time in seconds from the trigger to the display reference
          in NR3 format
```

The :TIMebase[:MAIN]: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[:MAIN]:DELay command.

Query Syntax

```
:TIMebase[:MAIN]:POSition?
```

The :TIMebase[:MAIN]:POSition? query returns the current time from the trigger to the display reference in seconds.

Return Format

```
<pos><NL>

<pos> ::= time in seconds from the trigger to the display reference
          in NR3 format
```

See Also

- "Introduction to :TIMebase Commands" on page 1066
- ":TIMebase:REFerence" on page 1071
- ":TIMebase[:MAIN]:RANGe" on page 1068
- ":TIMebase[:MAIN]:SCALe" on page 1069
- ":TIMebase:WINDow:POSition" on page 1074
- ":TIMebase[:MAIN]:DELay" on page 1243

:TIMebase[:MAIN]:RANGe

 (see page 1292)

Command Syntax

```
:TIMebase[:MAIN]:RANGe <range_value>

<range_value> ::= time for 10 div in seconds in NR3 format
```

The :TIMebase[:MAIN]: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[:MAIN]:RANGe?
```

The :TIMebase[:MAIN]:RANGe query returns the current full-scale range value for the main window.

Return Format

```
<range_value><NL>

<range_value> ::= time for 10 div in seconds in NR3 format
```

See Also

- "Introduction to :TIMebase Commands" on page 1066
- ":TIMebase:MODE" on page 1070
- ":TIMebase[:MAIN]:SCALe" on page 1069
- ":TIMebase:WINDow:RANGe" on page 1075

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.
```

See complete example programs at: Chapter 44, “Programming Examples,” starting on page 1301

:TIMebase[:MAIN]:SCALe

 (see page 1292)

Command Syntax

```
:TIMebase[:MAIN]:SCALe <scale_value>

<scale_value> ::= time/div in seconds in NR3 format
```

The :TIMebase[:MAIN]:SCALe command sets the horizontal scale or units per division for the main window.

Query Syntax

```
:TIMebase[:MAIN]:SCALe?
```

The :TIMebase[:MAIN]:SCALe? query returns the current horizontal scale setting in seconds per division for the main window.

Return Format

```
<scale_value><NL>

<scale_value> ::= time/div in seconds in NR3 format
```

See Also

- "Introduction to :TIMebase Commands" on page 1066
- ":TIMebase[:MAIN]:RANGe" on page 1068
- ":TIMebase:WINDow:SCALe" on page 1076
- ":TIMebase:WINDow:RANGe" on page 1075

:TIMebase:MODE

(see page 1292)

Command Syntax

```
:TIMebase:MODE <value>

<value> ::= {MAIN | WINDow}
```

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 (zoomed or delayed) time base mode, measurements are made in the zoomed time base if possible; otherwise, the measurements are made in the main time base.

Query Syntax

```
:TIMebase:MODE?
```

The :TIMebase:MODE query returns the current time base mode.

Return Format

```
<value><NL>

<value> ::= {MAIN | WIND}
```

See Also

- "Introduction to :TIMebase Commands" on page 1066
- "*RST (Reset)" on page 194
- ":TIMebase[:MAIN]:RANGe" on page 1068
- ":TIMebase[:MAIN]:POSition" on page 1067
- ":TIMebase:REFerence" on page 1071

Example Code

```
' TIMEBASE_MODE - (not executed in this example)
' Set the time base mode to MAIN, DELAYED.

' Set time base mode to main.
myScope.WriteString ":TIMebase:MODE MAIN"
```

See complete example programs at: Chapter 44, “Programming Examples,” starting on page 1301

:TIMebase:REFerence

(see page 1292)

Command Syntax

```
:TIMebase:REFerence <reference>

<reference> ::= {LEFT | CENTer | RIGHt | CUSTom}
```

The :TIMebase:REFerence command sets the time reference to:

- LEFT – one division from the left side of the screen.
- CENTer – the center of the screen.
- RIGHt – one division from the right side of the screen.
- CUSTom – lets you use the :TIMebase:REFerence:LOCation command to place the time reference location at a percent of the graticule width (where 0.0 is the left edge and 1.0 is the right edge).

The 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 | CUST}
```

See Also

- "Introduction to :TIMebase Commands" on page 1066
- ":TIMebase:REFerence:LOCation" on page 1072
- ":TIMebase:MODE" on page 1070

Example Code

```
myScope.WriteString ":TIMebase:REFerence CENTer"    ' Set reference to
center.
```

See complete example programs at: Chapter 44, “Programming Examples,” starting on page 1301

:TIMebase:REFerence:LOCation

(see page 1292)

Command Syntax

```
:TIMebase:REFerence:LOCation <loc>

<loc> ::= 0.0 to 1.0 in NR3 format
```

When the :TIMebase:REFerence is set to CUSTom, the :TIMebase:REFerence:LOCation command lets you place the time reference location at a percent of the graticule width (where 0.0 is the left edge and 1.0 is the right edge).

Query Syntax

```
:TIMebase:REFerence:LOCation?
```

The :TIMebase:REFerence:LOCation? query returns the time base reference custom location setting.

Return Format

```
<loc><NL>

<loc> ::= 0.0 to 1.0 in NR3 format
```

See Also

- ":TIMebase:REFerence" on page 1071

:TIMebase:VERNier

 (see page 1292)

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" on page 1066

:TIMebase:WINDow:POSition

 (see page 1292)

Command Syntax

```
:TIMebase:WINDow:POSition <pos value>

<pos value> ::= time from the trigger event to the zoomed (delayed)
                view reference point in NR3 format
```

The :TIMebase:WINDow:POSition command sets the horizontal position in the zoomed (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 zoomed 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 zoomed view.

Return Format

```
<value><NL>

<value> ::= position value in seconds
```

See Also

- "Introduction to :TIMebase Commands" on page 1066
- ":TIMebase:MODE" on page 1070
- ":TIMebase[:MAIN]:POSition" on page 1067
- ":TIMebase[:MAIN]:RANGe" on page 1068
- ":TIMebase[:MAIN]:SCALe" on page 1069
- ":TIMebase:WINDow:RANGe" on page 1075
- ":TIMebase:WINDow:SCALe" on page 1076

:TIMebase:WINDow:RANGe

(see page 1292)

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 zoomed (delayed) window. The range is 10 times the current zoomed 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" on page 1066
- ":TIMebase[:MAIN]:RANGe" on page 1068
- ":TIMebase[:MAIN]:POSition" on page 1067
- ":TIMebase[:MAIN]:SCALe" on page 1069

:TIMebase:WINDow:SCALe

(see page 1292)

Command Syntax

```
:TIMebase:WINDow:SCALe <scale_value>

<scale_value> ::= scale value in seconds in NR3 format
```

The :TIMebase:WINDow:SCALe command sets the zoomed (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 zoomed window scale setting.

Return Format

```
<scale_value><NL>

<scale_value> ::= current seconds per division for the zoomed window
```

See Also

- "Introduction to :TIMebase Commands" on page 1066
- ":TIMebase[:MAIN]:RANGe" on page 1068
- ":TIMebase[:MAIN]:POSition" on page 1067
- ":TIMebase[:MAIN]:SCALe" on page 1069
- ":TIMebase:WINDow:RANGe" on page 1075

35 :TRIGger Commands

Control the trigger modes and parameters for each trigger type. See:

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 ":TRIGger:SWEep" on page 1093) 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 ":TRIGger:MODE" on page 1091).

- **Edge triggering** — identifies a trigger by looking for a specified slope and voltage level on a waveform.

- **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. You can also trigger on a specified time duration of a pattern.

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 +4.00000000E-08;
:TRIG:EDGE:SOUR CHAN1;LEV +0.0E+00;SLOP POS;REJ OFF;COUP DC;
:TRIG:ZONE1:STAT 0;:TRIG:ZONE2:STAT 0
```

General :TRIGger Commands

Table 134 General :TRIGger Commands Summary

Command	Query	Options and Query Returns
:TRIGger:FORCe (see page 1081)	n/a	n/a
:TRIGger:HFReject {{0 | OFF} | {1 | ON}} (see page 1082)	:TRIGger:HFReject? (see page 1082)	{0 | 1}
:TRIGger:HOLDoff <holdoff_time> (see page 1083)	:TRIGger:HOLDoff? (see page 1083)	<holdoff_time> ::= 60 ns to 10 s in NR3 format
:TRIGger:HOLDoff:MAXimum <max_holdoff> (see page 1084)	:TRIGger:HOLDoff:MAXimum? (see page 1084)	<max_holdoff> ::= maximum holdoff time in seconds in NR3 format
:TRIGger:HOLDoff:MINimum <min_holdoff> (see page 1085)	:TRIGger:HOLDoff:MINimum? (see page 1085)	<min_holdoff> ::= minimum holdoff time in seconds in NR3 format
:TRIGger:HOLDoff:RANDom {{0 | OFF} | {1 | ON}} (see page 1086)	:TRIGger:HOLDoff:RANDom? (see page 1086)	<setting> ::= {0 | 1}
:TRIGger:JFRee {{0 | OFF} | {1 | ON}} (see page 1087)	:TRIGger:JFRee? (see page 1087)	{0 | 1}
:TRIGger:LEVel:ASETup (see page 1088)	n/a	n/a
:TRIGger:LEVel:HIGH <level>, <source> (see page 1089)	:TRIGger:LEVel:HIGH? <source> (see page 1089)	<level> ::= .75 x full-scale voltage from center screen in NR3 format. <source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:LEVel:LOW <level>, <source> (see page 1090)	:TRIGger:LEVel:LOW? <source> (see page 1090)	<level> ::= .75 x full-scale voltage from center screen in NR3 format. <source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format

Table 134 General :TRIGger Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:MODE <mode> (see page 1091)	:TRIGger:MODE? (see page 1091)	<mode> ::= {EDGE | GLITch | PATTern | DELay | OR | RUNT | SHOLd | TRANsition | SBUS{1 | 2}} <return_value> ::= {<mode>}
:TRIGger:NREJect {{0 | OFF} | {1 | ON}} (see page 1092)	:TRIGger:NREJect? (see page 1092)	{0 | 1}
:TRIGger:SWEep <sweep> (see page 1093)	:TRIGger:SWEep? (see page 1093)	<sweep> ::= {AUTO | NORMal}

:TRIGger:FORCe

 (see page 1292)

Command Syntax `:TRIGger:FORCe`

The :TRIGger:FORCe command causes an acquisition to be captured even though the trigger condition has not been met. This command is equivalent to the front panel **[Force Trigger]** key.

See Also

- "Introduction to :TRIGger Commands" on page 1077

:TRIGger:HFReject

(see page 1292)

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" on page 1077
- ":TRIGger[:EDGE]:REJect" on page 1097

:TRIGger:HOLDoff

(see page 1292)

Command Syntax

```
:TRIGger:HOLDoff <holdoff_time>

<holdoff_time> ::= 40 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" on page 1077

:TRIGger:HOLDoff:MAXimum

N (see page 1292)

Command Syntax

```
:TRIGger:HOLDoff:MAXimum <max_holdoff>

<max_holdoff> ::= maximum holdoff time in seconds in NR3 format
```

When the random trigger holdoff mode is enabled (see :TRIGger:HOLDoff:RANDom), the :TRIGger:HOLDoff:MAXimum command specifies the maximum trigger holdoff time.

Query Syntax

```
:TRIGger:HOLDoff:MAXimum?
```

The :TRIGger:HOLDoff:MAXimum? query returns the maximum random trigger holdoff time setting.

Return Format

```
<max_holdoff><NL>
```

See Also

- ":TRIGger:HOLDoff:MINimum" on page 1085
- ":TRIGger:HOLDoff:RANDom" on page 1086

:TRIGger:HOLDoff:MINimum

N (see page 1292)

Command Syntax

```
:TRIGger:HOLDoff:MINimum <min_holdoff>

<min_holdoff> ::= minimum holdoff time in seconds in NR3 format
```

When the random trigger holdoff mode is enabled (see :TRIGger:HOLDoff:RANDom), the :TRIGger:HOLDoff:MINimum command specifies the minimum trigger holdoff time.

Query Syntax

```
:TRIGger:HOLDoff:MINimum?
```

The :TRIGger:HOLDoff:MINimum? query returns the minimum random trigger holdoff time setting.

Return Format

```
<min_holdoff><NL>
```

See Also

- ":TRIGger:HOLDoff:MAXimum" on page 1084
- ":TRIGger:HOLDoff:RANDom" on page 1086

:TRIGger:HOLDoff:RANDom

(see page 1292)

Command Syntax `:TRIGger:HOLDoff:RANDom {{0 | OFF} | {1 | ON}}`

The :TRIGger:HOLDoff:RANDom command enables or disables the random trigger holdoff mode. This mode randomizes the holdoff time from one acquisition to the next. The randomized holdoff time values will be between the values specified by the :TRIGger:HOLDoff:MINimum and :TRIGger:HOLDoff:MAXimum commands.

The random trigger holdoff mode ensures that the oscilloscope re-arms after each acquisition in a manner that minimizes or eliminates the likelihood of triggering at the beginning of a DDR burst. Randomizing the holdoff time increases the likelihood that the oscilloscope will trigger on different data phases of a multi-phase (8 data transfer) burst. This mode mixes up the traffic pattern the oscilloscope triggers on and is very effective when used on repeating patterns.

Query Syntax `:TRIGger:HOLDoff:RANDom?`

The :TRIGger:HOLDoff:RANDom? query returns random trigger holdoff mode setting.

Return Format `<setting><NL>`

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

See Also

- ":TRIGger:HOLDoff:MAXimum" on page 1084
- ":TRIGger:HOLDoff:MINimum" on page 1085

:TRIGger:JFRee

 (see page 1292)

Command Syntax `:TRIGger:JFRee {{0 | OFF} | {1 | ON}}`

The :TRIGger:JFRee command disables or enables the Jitter-Free Trigger option.

Oscilloscope trigger circuitry introduces a certain amount of noise and jitter that result in horizontal error. To correct for this error, in certain trigger modes and at certain timebase ranges, hardware Jitter-Free Trigger correction is used to minimize trigger jitter at the trigger point.

The effect seen on the oscilloscope display is that all plotted acquisitions cross through the same pixel location at the intersection of the trigger level and time=0.

Jitter-Free Trigger correction works well for fast digital edges, but you may want to turn it off for some slow slew rate analog signals.

Query Syntax `:TRIGger:JFRee?`

The :TRIGger:JFRee? query returns the Jitter-Free Trigger option setting.

Return Format `{0 | 1}`

:TRIGger:LEVel:ASETup

(see page 1292)

Command Syntax `:TRIGger:LEVel:ASETup`

The :TRIGger:LEVel:ASETup command automatically sets the trigger levels of all displayed analog channels to their waveforms' 50% values.

If AC coupling is used, the trigger levels are set to 0 V.

When High and Low (dual) trigger levels are used (as with Rise/Fall Time and Runt triggers, for example), this command has no effect.

See Also

- ":TRIGger[:EDGE]:LEVel" on page 1096

:TRIGger:LEVel:HIGH

N (see page 1292)

Command Syntax

```
:TRIGger:LEVel:HIGH <level>, <source>

<level> ::= 0.75 x full-scale voltage from center screen in NR3 format
            for internal triggers

<source> ::= CHANnel<n>

<n> ::= 1 to (# analog channels) in NR1 format
```

The :TRIGger:LEVel:HIGH command sets the high trigger voltage level voltage for the specified source.

High and low trigger levels are used with runt triggers and rise/fall time (transition) triggers.

Query Syntax

```
:TRIGger:LEVel:HIGH? <source>
```

The :TRIGger:LEVel:HIGH? query returns the high trigger voltage level for the specified source.

Return Format

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

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:LEVel:LOW" on page 1090
- ":TRIGger:RUNT Commands" on page 1119
- ":TRIGger:TRANsition Commands" on page 1130
- ":TRIGger[:EDGE]:SOURce" on page 1099

:TRIGger:LEVel:LOW

(see page 1292)

Command Syntax

```
:TRIGger:LEVel:LOW <level>, <source>

<level> ::= 0.75 x full-scale voltage from center screen in NR3 format
            for internal triggers

<source> ::= CHANnel<n>

<n> ::= 1 to (# analog channels) in NR1 format
```

The :TRIGger:LEVel:LOW command sets the low trigger voltage level voltage for the specified source.

High and low trigger levels are used with runt triggers and rise/fall time (transition) triggers.

Query Syntax

```
:TRIGger:LEVel:LOW? <source>
```

The :TRIGger:LEVel:LOW? query returns the low trigger voltage level for the specified source.

Return Format

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

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:LEVel:HIGH" on page 1089
- ":TRIGger:RUNT Commands" on page 1119
- ":TRIGger:TRANsition Commands" on page 1130
- ":TRIGger[:EDGE]:SOURce" on page 1099

:TRIGger:MODE

(see page 1292)

Command Syntax

```
:TRIGger:MODE <mode>

<mode> ::= {EDGE | GLITch | PATTern | OR | RUNT
           | SHOLd | TRANsition | SBUS{1 | 2}}
```

The :TRIGger:MODE command selects the trigger mode (trigger type).

Query Syntax

```
:TRIGger:MODE?
```

The :TRIGger:MODE? query returns the current trigger mode.

Return Format

```
<mode><NL>

<mode> ::= {EDGE | GLIT | PATT | OR | RUNT | SHOL
           | TRAN | SBUS{1 | 2}}
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:SWEep" on page 1093
- ":TIMebase:MODE" on page 1070

Example Code

```
' TRIGGER_MODE - Set the trigger mode to EDGE.
myScope.WriteString ":TRIGger:MODE EDGE"
```

See complete example programs at: Chapter 44, “Programming Examples,” starting on page 1301

:TRIGger:NREJect

 (see page 1292)

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.

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" on page 1077

:TRIGger:SWEep

 (see page 1292)

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" on page 1077

:TRIGger[:EDGE] Commands

Table 135 :TRIGger[:EDGE] Commands Summary

Command	Query	Options and Query Returns
:TRIGger[:EDGE]:COUPling {AC | DC | LFReject} (see page 1095)	:TRIGger[:EDGE]:COUPling? (see page 1095)	{AC | DC | LFReject}
:TRIGger[:EDGE]:LEVel <level> [,<source>] (see page 1096)	:TRIGger[:EDGE]:LEVel? [<source>] (see page 1096)	For internal triggers, <level> ::= .75 x full-scale voltage from center screen in NR3 format. For external triggers, <level> ::= ±(external range setting) in NR3 format. For digital channels, <level> ::= ±8 V. <source> ::= {CHANnel<n> | DIGital<d> | EXTernal } <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:TRIGger[:EDGE]:REJect {OFF | LFReject | HFReject} (see page 1097)	:TRIGger[:EDGE]:REJect? (see page 1097)	{OFF | LFReject | HFReject}
:TRIGger[:EDGE]:SLOPe <polarity> (see page 1098)	:TRIGger[:EDGE]:SLOPe? (see page 1098)	<polarity> ::= {POSitive | NEGative | EITHer | ALTernate}
:TRIGger[:EDGE]:SOURce <source> (see page 1099)	:TRIGger[:EDGE]:SOURce? (see page 1099)	<source> ::= {CHANnel<n> | DIGital<d> | EXTernal | LINE | WGEN | WGEN1 | WMOD} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format Note: WGEN and WGEN1 are equivalent.

:TRIGger[:EDGE]:COUPling

(see page 1292)

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 LFReject.

- 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. Use AC coupling to get a stable edge trigger when your waveform has a large dc offset.
- LFReject coupling places a 50 kHz high-pass filter in the trigger path.
- 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" on page 1077
- ":TRIGger:MODE" on page 1091
- ":TRIGger[:EDGE]:REJect" on page 1097

:TRIGger[:EDGE]:LEVel

(see page 1292)

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> ::= ±(external range setting) in NR3 format
            for external triggers

<level> ::= ±8 V for digital channels

<source> ::= {CHANnel<n> | DIGital<d> | EXTernal}

<n> ::= 1 to (# analog channels) in NR1 format

<d> ::= 0 to (# digital channels - 1) in NR1 format
```

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" on page 1077
- ":TRIGger[:EDGE]:SOURce" on page 1099
- ":EXTernal:RANGe" on page 372
- ":POD<n>:THReshold" on page 673
- ":DIGital<d>:THReshold" on page 318

:TRIGger[:EDGE]:REJect

(see page 1292)

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" on page 1077
- ":TRIGger:HFReject" on page 1082
- ":TRIGger[:EDGE]:COUPling" on page 1095

:TRIGger[:EDGE]:SLOPe

(see page 1292)

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.

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" on page 1077
- ":TRIGger:MODE" on page 1091

Example Code

```
' TRIGGER_EDGE_SLOPE - Sets the slope of the edge for the trigger.

' Set the slope to positive.
myScope.WriteString ":TRIGGER:EDGE:SLOPE POSITIVE"
```

See complete example programs at: Chapter 44, “Programming Examples,” starting on page 1301

:TRIGger[:EDGE]:SOURce

(see page 1292)

Command Syntax

```
:TRIGger[:EDGE]:SOURce <source>

<source> ::= {CHANnel<n> | DIGital<d> | EXTernal | LINE | WGEN | WGEN1
             | WMOD}

<n> ::= 1 to (# analog channels) in NR1 format

<d> ::= 0 to (# digital channels - 1) in NR1 format

Note: WAVE and WGEN1 are equivalent.
```

The :TRIGger[:EDGE]:SOURce command selects the input that produces the trigger.

- EXTernal – triggers on the rear panel EXT TRIG IN signal.
- LINE – triggers at the 50% level of the rising or falling edge of the AC power source signal.
- WGEN, WGEN1 – triggers at the 50% level of the rising edge of the waveform generator output signal. This option is not available when the DC, NOISe, or CARDiac waveforms are selected.
- WMOD – when waveform generator FM modulation is used, triggers at the 50% level of the rising edge of the modulating signal.

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> | DIG<d> | EXTernal | LINE | WGEN | WGEN1
             | WMOD | NONE}
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:MODE" on page 1091

Example Code

```
   ' TRIGGER_EDGE_SOURCE - Selects the channel that actually produces th
e
   ' edge trigger.  Any channel can be selected.
   myScope.WriteString ":TRIGger:EDGE:SOURce CHANnel1"
```

See complete example programs at: Chapter 44, “Programming Examples,” starting on page 1301

:TRIGger:GLITch Commands

Table 136 :TRIGger:GLITch Commands Summary

Command	Query	Options and Query Returns
:TRIGger:GLITch:GREaterthan <greater_than_time>[suffix] (see page 1102)	:TRIGger:GLITch:GREaterthan? (see page 1102)	<greater_than_time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:GLITch:LESSthan <less_than_time>[suffix] (see page 1103)	:TRIGger:GLITch:LESSthan? (see page 1103)	<less_than_time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:GLITch:LEVel <level> [<source>] (see page 1104)	:TRIGger:GLITch:LEVel? (see page 1104)	For internal triggers, <level> ::= .75 x full-scale voltage from center screen in NR3 format. For digital channels, <level> ::= ±8 V. <source> ::= {CHANnel<n> | DIGital<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:TRIGger:GLITch:POLarity <polarity> (see page 1105)	:TRIGger:GLITch:POLarity? (see page 1105)	<polarity> ::= {POSitive | NEGative}
:TRIGger:GLITch:QUALifier <qualifier> (see page 1106)	:TRIGger:GLITch:QUALifier? (see page 1106)	<qualifier> ::= {GREaterthan | LESSthan | RANGe}

Table 136 :TRIGger:GLITch Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:GLITch:RANGe <less_than_time>[suffix], <greater_than_time>[suffix] (see page 1107)	:TRIGger:GLITch:RANGe? (see page 1107)	<less_than_time> ::= 15 ns to 10 seconds in NR3 format <greater_than_time> ::= 10 ns to 9.99 seconds in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:GLITch:SOURce <source> (see page 1108)	:TRIGger:GLITch:SOURce? (see page 1108)	<source> ::= {CHANnel<n> | DIGital<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format

:TRIGger:GLITch:GREaterthan

(see page 1292)

Command Syntax

```
:TRIGger:GLITch:GREaterthan <greater_than_time>[<suffix>]

<greater_than_time> ::= floating-point number 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>

<greater_than_time> ::= floating-point number in NR3 format.
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:GLITch:SOURce" on page 1108
- ":TRIGger:GLITch:QUALifier" on page 1106
- ":TRIGger:MODE" on page 1091

:TRIGger:GLITch:LESSthan

N (see page 1292)

Command Syntax

```
:TRIGger:GLITch:LESSthan <less_than_time>[<suffix>]

<less_than_time> ::= floating-point number in NR3 format

<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> ::= floating-point number in NR3 format.
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:GLITch:SOURce" on page 1108
- ":TRIGger:GLITch:QUALifier" on page 1106
- ":TRIGger:MODE" on page 1091

:TRIGger:GLITch:LEVel

(see page 1292)

Command Syntax

```
:TRIGger:GLITch:LEVel <level>[,<source>]

<level> ::= .75 x full-scale voltage from center screen in NR3 format
            for internal triggers

<level> ::= ±8 V for digital channels

<source> ::= {CHANnel<n> | DIGital<d>}

<n> ::= 1 to (# analog channels) in NR1 format

<d> ::= 0 to (# digital channels - 1) in NR1 format
```

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><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:MODE" on page 1091
- ":TRIGger:GLITch:SOURce" on page 1108
- ":EXTernal:RANGe" on page 372

:TRIGger:GLITch:POLarity

 (see page 1292)

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" on page 1077
- ":TRIGger:MODE" on page 1091
- ":TRIGger:GLITch:SOURce" on page 1108

:TRIGger:GLITch:QUALifier

N (see page 1292)

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" on page 1077
- ":TRIGger:GLITch:SOURce" on page 1108
- ":TRIGger:MODE" on page 1091

:TRIGger:GLITch:RANGe

(see page 1292)

Command Syntax

```
:TRIGger:GLITch:RANGe <less_than_time>[suffix],
                      <greater_than_time>[suffix]

<less_than_time> ::= (15 ns - 10 seconds) in NR3 format

<greater_than_time> ::= (10 ns - 9.99 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. You can enter the parameters in any order — the smaller value becomes the <greater_than_time> and the larger value becomes the <less_than_time>.

Query Syntax

```
:TRIGger:GLITch:RANGe?
```

The :TRIGger:GLITch:RANGe? query returns the pulse width duration time for :TRIGger:GLITch:SOURce.

Return Format

```
<less_than_time>,<greater_than_time><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:GLITch:SOURce" on page 1108
- ":TRIGger:GLITch:QUALifier" on page 1106
- ":TRIGger:MODE" on page 1091

:TRIGger:GLITch:SOURce

(see page 1292)

Command Syntax

```
:TRIGger:GLITch:SOURce <source>

<source> ::= {DIGital<d> | CHANnel<n>}

<n> ::= 1 to (# analog channels) in NR1 format

<d> ::= 0 to (# digital channels - 1) in NR1 format
```

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" on page 1077
- ":TRIGger:MODE" on page 1091
- ":TRIGger:GLITch:LEVel" on page 1104
- ":TRIGger:GLITch:POLarity" on page 1105
- ":TRIGger:GLITch:QUALifier" on page 1106
- ":TRIGger:GLITch:RANGe" on page 1107

Example Code

- "Example Code" on page 1099

:TRIGger:OR Commands

Table 137 :TRIGger:OR Commands Summary

Command	Query	Options and Query Returns
:TRIGger:OR <string> (see page 1110)	:TRIGger:OR? (see page 1110)	<string> ::= "nn...n" where n ::= {R | F | E | X} R = rising edge, F = falling edge, E = either edge, X = don't care. Each character in the string is for an analog or digital channel as shown on the front panel display.

:TRIGger:OR

(see page 1292)

Command Syntax

```
:TRIGger:OR <string>

<string> ::= "nn...n" where n ::= {R | F | E | X}

R = rising edge, F = falling edge, E = either edge, X = don't care.
```

The :TRIGger:OR command specifies the edges to include in the OR'ed edge trigger.

In the <string> parameter, each bit corresponds to a channel as described in the following table:

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 4 through 1.
2 analog + 16 digital channels (mixed-signal)	Bits 0 through 15 - digital channels 0 through 15. Bits 16 and 17 - analog channels 2 and 1.

Query Syntax

```
:TRIGger:OR?
```

The :TRIGger:OR? query returns the current OR'ed edge trigger string.

Return Format

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

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:MODE" on page 1091

:TRIGger:PATTern Commands

Table 138 :TRIGger:PATTern Commands Summary

Command	Query	Options and Query Returns
:TRIGger:PATTern <string>[,<edge_source>,<edge>] (see page 1112)	:TRIGger:PATTern? (see page 1113)	<string> ::= "nn...n" where n ::= {0 | 1 | X | R | F} when <base> = ASCii <string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $} when <base> = HEX <edge_source> ::= {CHANnel<n> | DIGital<d> | NONE} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <edge> ::= {POSitive | NEGative}
:TRIGger:PATTern:FORMat <base> (see page 1114)	:TRIGger:PATTern:FORMat? (see page 1114)	<base> ::= {ASCii | HEX}
:TRIGger:PATTern:GREaterthan <greater_than_time>[suffix] (see page 1115)	:TRIGger:PATTern:GREaterthan? (see page 1115)	<greater_than_time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:PATTern:LESSthan <less_than_time>[suffix] (see page 1116)	:TRIGger:PATTern:LESSthan? (see page 1116)	<less_than_time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:PATTern:QUALifier <qualifier> (see page 1117)	:TRIGger:PATTern:QUALifier? (see page 1117)	<qualifier> ::= {ENTered | GREaterthan | LESSthan | INRange | OUTRange | TIMeout}
:TRIGger:PATTern:RANGe <less_than_time>[suffix], <greater_than_time>[suffix] (see page 1118)	:TRIGger:PATTern:RANGe? (see page 1118)	<less_than_time> ::= 15 ns to 10 seconds in NR3 format <greater_than_time> ::= 10 ns to 9.99 seconds in NR3 format [suffix] ::= {s | ms | us | ns | ps}

:TRIGger:PATTern

(see page 1292)

Command Syntax

```
:TRIGger:PATTern <pattern>

<pattern> ::= <string>[,<edge_source>,<edge>]

<string> ::= "nn...n" where n ::= {0 | 1 | X | R | F} when
             <base> = ASCii

<string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $} when
             <base> = HEX

<edge_source> ::= {CHANnel<n> | DIGital<d>
                  | NONE}

<n> ::= 1 to (# of analog channels) in NR1 format

<d> ::= 0 to (# digital channels - 1) in NR1 format

<edge> ::= {POSitive | NEGative}
```

The :TRIGger:PATTern command specifies the channel values to be used in the pattern trigger.

In the <string> parameter, each bit corresponds to a channel as described in the following table:

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 4 through 1.
2 analog + 16 digital channels (mixed-signal)	Bits 0 through 15 - digital channels 0 through 15. Bits 16 and 17 - analog channels 2 and 1.

The format of the <string> parameter depends on the :TRIGger:PATTern:FORMat command setting:

- When the format is ASCii, the string looks just like the string you see on the oscilloscope's front panel, made up of 0, 1, X (don't care), R (rising edge), and F (falling edge) characters.
- When the format is HEX, the string begins with "0x" and contains hex digit characters or X (don't care for all four bits in the nibble).

  With the hex format string, you can use the <edge_source> and <edge> parameters to specify an edge on one of the channels.

NOTE The optional <edge_source> and <edge> parameters should be sent together or not at all. The edge can be specified in the ASCII <string> parameter. If the edge source and edge parameters are used, they take precedence.

You can only specify an edge on one channel. When an edge is specified, the :TRIGger:PATTern:QUALifier does not apply.

Query Syntax

```
:TRIGger:PATTern?
```

The :TRIGger:PATTern? query returns the pattern string, edge source, and edge.

Return Format

```
<string>,<edge_source>,<edge><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:PATTern:FORMat" on page 1114
- ":TRIGger:PATTern:QUALifier" on page 1117
- ":TRIGger:MODE" on page 1091
- "Commands Not Supported in Multiple Program Message Units" on page 1297

:TRIGger:PATTern:FORMat

(see page 1292)

Command Syntax

```
:TRIGger:PATTern:FORMat <base>

<base> ::= {ASCii | HEX}
```

The :TRIGger:PATTern:FORMat command sets the entry (and query) number base used by the :TRIGger:PATTern command. The default <base> is ASCii.

Query Syntax

```
:TRIGger:PATTern:FORMat?
```

The :TRIGger:PATTern:FORMat? query returns the currently set number base for pattern trigger patterns.

Return Format

```
<base><NL>

<base> ::= {ASC | HEX}
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:PATTern" on page 1112

:TRIGger:PATTern:GREaterthan

(see page 1292)

Command Syntax

```
:TRIGger:PATTern:GREaterthan <greater_than_time>[<suffix>]

<greater_than_time> ::= minimum trigger duration in seconds
                        in NR3 format

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

The :TRIGger:PATTern:GREaterthan command sets the minimum duration for the defined pattern when :TRIGger:PATTern:QUALifier is set to GREaterthan. The command also sets the timeout value when the :TRIGger:PATTern:QUALifier is set to TIMeout.

Query Syntax

```
:TRIGger:PATTern:GREaterthan?
```

The :TRIGger:PATTern:GREaterthan? query returns the minimum duration time for the defined pattern.

Return Format

```
<greater_than_time><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:PATTern" on page 1112
- ":TRIGger:PATTern:QUALifier" on page 1117
- ":TRIGger:MODE" on page 1091

:TRIGger:PATTern:LESSthan

N (see page 1292)

Command Syntax

```
:TRIGger:PATTern:LESSthan <less_than_time>[<suffix>]

<less_than_time> ::= maximum trigger duration in seconds
                     in NR3 format

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

The :TRIGger:PATTern:LESSthan command sets the maximum duration for the defined pattern when :TRIGger:PATTern:QUALifier is set to LESSthan.

Query Syntax

```
:TRIGger:PATTern:LESSthan?
```

The :TRIGger:PATTern:LESSthan? query returns the duration time for the defined pattern.

Return Format

```
<less_than_time><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:PATTern" on page 1112
- ":TRIGger:PATTern:QUALifier" on page 1117
- ":TRIGger:MODE" on page 1091

:TRIGger:PATTern:QUALifier

N (see page 1292)

Command Syntax

```
:TRIGger:PATTern:QUALifier <qualifier>

<qualifier> ::= {ENTered | GREaterthan | LESSthan | INRange | OUTRange
                | TIMeout}
```

The :TRIGger:PATTern:QUALifier command qualifies when the trigger occurs:

- ENTered – when the pattern is entered.
- LESSthan – when the pattern is present for less than a time value.
- GREaterthan – when the pattern is present for greater than a time value. The trigger occurs when the pattern exits (not when the GREaterthan time value is exceeded).
- TIMeout – when the pattern is present for greater than a time value. In this case, the trigger occurs when the GREaterthan time value is exceeded (not when the pattern exits).
- INRange – when the pattern is present for a time within a range of values.
- OUTRange – when the pattern is present for a time outside of range of values.

Pattern durations are evaluated using a timer. The timer starts on the last edge that makes the pattern (logical AND) true. Except when the TIMeout qualifier is selected, the trigger occurs on the first edge that makes the pattern false, provided the time qualifier criteria has been met.

Set the GREaterthan qualifier value with the :TRIGger:PATTern:GREaterthan command.

Set the LESSthan qualifier value with the :TRIGger:PATTern:LESSthan command.

Set the INRange and OUTRange qualifier values with the :TRIGger:PATTern:RANGe command.

Set the TIMeout qualifier value with the :TRIGger:PATTern:GREaterthan command.

Query Syntax

```
:TRIGger:PATTern:QUALifier?
```

The :TRIGger:PATTern:QUALifier? query returns the trigger duration qualifier.

Return Format

```
<qualifier><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:PATTern:GREaterthan" on page 1115
- ":TRIGger:PATTern:LESSthan" on page 1116
- ":TRIGger:PATTern:RANGe" on page 1118

:TRIGger:PATTern:RANGe

(see page 1292)

Command Syntax

```
:TRIGger:PATTern:RANGe <less_than_time>[<suffix>],
                       <greater_than_time>[<suffix>]

<greater_than_time> ::= 10 ns to 9.99 seconds in NR3 format

<less_than_time> ::= 15 ns to 10 seconds in NR3 format

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

The :TRIGger:PATTern:RANGe command sets the duration for the defined pattern when the :TRIGger:PATTern:QUALifier command is set to INRange or OUTRange. You can enter the parameters in any order – the smaller value becomes the <greater_than_time> and the larger value becomes the <less_than_time>.

Query Syntax

```
:TRIGger:PATTern:RANGe?
```

The :TRIGger:PATTern:RANGe? query returns the duration time for the defined pattern.

Return Format

```
<less_than_time>,<greater_than_time><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:PATTern" on page 1112
- ":TRIGger:PATTern:QUALifier" on page 1117
- ":TRIGger:MODE" on page 1091

:TRIGger:RUNT Commands

Table 139 :TRIGger:RUNT Commands Summary

Command	Query	Options and Query Returns
:TRIGger:RUNT:POLarity <polarity> (see page 1120)	:TRIGger:RUNT:POLarity? (see page 1120)	<polarity> ::= {POSitive | NEGative}
:TRIGger:RUNT:QUALifier <qualifier> (see page 1121)	:TRIGger:RUNT:QUALifier? (see page 1121)	<qualifier> ::= {GREaterthan | LESSthan | NONE}
:TRIGger:RUNT:SOURce <source> (see page 1122)	:TRIGger:RUNT:SOURce? (see page 1122)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:RUNT:TIME <time>[suffix] (see page 1123)	:TRIGger:RUNT:TIME? (see page 1123)	<time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}

:TRIGger:RUNT:POLarity

(see page 1292)

Command Syntax

```
:TRIGger:RUNT:POLarity <polarity>

<polarity> ::= {POSitive | NEGative}
```

The :TRIGger:RUNT:POLarity command sets the polarity for the runt trigger:

- POSitive – positive runt pulses.
- NEGative – negative runt pulses.

Query Syntax

```
:TRIGger:RUNT:POLarity?
```

The :TRIGger:RUNT:POLarity? query returns the runt trigger polarity.

Return Format

```
<polarity><NL>

<polarity> ::= {POS | NEG | EITH}
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:MODE" on page 1091
- ":TRIGger:LEVel:HIGH" on page 1089
- ":TRIGger:LEVel:LOW" on page 1090
- ":TRIGger:RUNT:SOURce" on page 1122

:TRIGger:RUNT:QUALifier

(see page 1292)

Command Syntax

```
:TRIGger:RUNT:QUALifier <qualifier>

<qualifier> ::= {GREaterthan | LESSthan | NONE}
```

The :TRIGger:RUNT:QUALifier command selects the qualifier used for specifying runt pulse widths:

- GREaterthan – triggers on runt pulses whose width is greater than the :TRIGger:RUNT:TIME.
- LESSthan – triggers on runt pulses whose width is less than the :TRIGger:RUNT:TIME.
- NONE – triggers on runt pulses of any width.

Query Syntax

```
:TRIGger:RUNT:QUALifier?
```

The :TRIGger:RUNT:QUALifier? query returns the runt trigger qualifier setting.

Return Format

```
<qualifier><NL>

<qualifier> ::= {GRE | LESS NONE}
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:MODE" on page 1091
- ":TRIGger:RUNT:TIME" on page 1123

:TRIGger:RUNT:SOURce

 (see page 1292)

Command Syntax

```
:TRIGger:RUNT:SOURce <source>

<source> ::= CHANnel<n>

<n> ::= 1 to (# analog channels) in NR1 format
```

The :TRIGger:RUNT:SOURce command selects the channel used to produce the trigger.

Query Syntax

```
:TRIGger:RUNT:SOURce?
```

The :TRIGger:RUNT:SOURce? query returns the current runt trigger source.

Return Format

```
<source><NL>

<source> ::= CHAN<n>
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:RUNT:POLarity" on page 1120

:TRIGger:RUNT:TIME

(see page 1292)

Command Syntax

```
:TRIGger:RUNT:TIME <time>[suffix]

<time> ::= floating-point number in NR3 format

[suffix] ::= {s | ms | us | ns | ps}
```

When triggering on runt pulses whose width is greater than or less than a certain value (see :TRIGger:RUNT:QUALifier), the :TRIGger:RUNT:TIME command specifies the time used with the qualifier.

Query Syntax

```
:TRIGger:RUNT:TIME?
```

The :TRIGger:RUNT:TIME? query returns the current runt pulse qualifier time setting.

Return Format

```
<time><NL>

<time> ::= floating-point number in NR3 format
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:RUNT:QUALifier" on page 1121

:TRIGger:SHOLd Commands

Table 140 :TRIGger:SHOLd Commands Summary

Command	Query	Options and Query Returns
:TRIGger:SHOLd:SLOPe <slope> (see page 1125)	:TRIGger:SHOLd:SLOPe? (see page 1125)	<slope> ::= {NEGative | POSitive}
:TRIGger:SHOLd:SOURce:CLOCk <source> (see page 1126)	:TRIGger:SHOLd:SOURce:CLOCk? (see page 1126)	<source> ::= {CHANnel<n> | DIGital<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:TRIGger:SHOLd:SOURce:DATA <source> (see page 1127)	:TRIGger:SHOLd:SOURce:DATA? (see page 1127)	<source> ::= {CHANnel<n> | DIGital<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:TRIGger:SHOLd:TIME:HOLD <time>[suffix] (see page 1128)	:TRIGger:SHOLd:TIME:HOLD? (see page 1128)	<time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:SHOLd:TIME:SETup <time>[suffix] (see page 1129)	:TRIGger:SHOLd:TIME:SETup? (see page 1129)	<time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}

:TRIGger:SHOLd:SLOPe

(see page 1292)

Command Syntax

```
:TRIGger:SHOLd:SLOPe <slope>

<slope> ::= {NEGative | POSitive}
```

The :TRIGger:SHOLd:SLOPe command specifies whether the rising edge or the falling edge of the clock signal is used.

Query Syntax

```
:TRIGger:SHOLd:SLOPe?
```

The :TRIGger:SHOLd:SLOPe? query returns the current rising or falling edge setting.

Return Format

```
<slope><NL>

<slope> ::= {NEG | POS}
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:MODE" on page 1091
- ":TRIGger:SHOLd:SOURce:CLOCk" on page 1126
- ":TRIGger:SHOLd:SOURce:DATA" on page 1127

:TRIGger:SHOLd:SOURce:CLOCk

N (see page 1292)

Command Syntax

```
:TRIGger:SHOLd:SOURce:CLOCk <source>

<source> ::= {CHANnel<n> | DIGital<d>}

<n> ::= 1 to (# analog channels) in NR1 format

<d> ::= 0 to (# digital channels - 1) in NR1 format
```

The :TRIGger:SHOLd:SOURce:CLOCk command selects the input channel probing the clock signal.

Query Syntax

```
:TRIGger:SHOLd:SOURce:CLOCk?
```

The :TRIGger:SHOLd:SOURce:CLOCk? query returns the currently set clock signal source.

Return Format

```
<source><NL>

<source> ::= {CHAN<n> | DIG<d>}
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:MODE" on page 1091
- ":TRIGger:SHOLd:SLOPe" on page 1125

:TRIGger:SHOLd:SOURce:DATA

N (see page 1292)

Command Syntax

```
:TRIGger:SHOLd:SOURce:DATA <source>

<source> ::= {CHANnel<n> | DIGital<d>}

<n> ::= 1 to (# analog channels) in NR1 format

<d> ::= 0 to (# digital channels - 1) in NR1 format
```

The :TRIGger:SHOLd:SOURce:DATA command selects the input channel probing the data signal.

Query Syntax

```
:TRIGger:SHOLd:SOURce:DATA?
```

The :TRIGger:SHOLd:SOURce:DATA? query returns the currently set data signal source.

Return Format

```
<source><NL>

<source> ::= {CHAN<n> | DIG<d>}
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:MODE" on page 1091
- ":TRIGger:SHOLd:SLOPe" on page 1125

:TRIGger:SHOLd:TIME:HOLD

(see page 1292)

Command Syntax

```
:TRIGger:SHOLd:TIME:HOLD <time>[suffix]
<time> ::= floating-point number in NR3 format
[suffix] ::= {s | ms | us | ns | ps}
```

The :TRIGger:SHOLd:TIME:HOLD command sets the hold time.

Query Syntax

```
:TRIGger:SHOLd:TIME:HOLD?
```

The :TRIGger:SHOLd:TIME:HOLD? query returns the currently specified hold time.

Return Format

```
<time><NL>
<time> ::= floating-point number in NR3 format
```

See Also

- "Introduction to :TRIGger Commands" on page 1077

:TRIGger:SHOLd:TIME:SETup

N (see page 1292)

Command Syntax

```
:TRIGger:SHOLd:TIME:SETup <time>[suffix]

<time> ::= floating-point number in NR3 format

[suffix] ::= {s | ms | us | ns | ps}
```

The :TRIGger:SHOLd:TIME:SETup command sets the setup time.

Query Syntax

```
:TRIGger:SHOLd:TIME:SETup?
```

The :TRIGger:SHOLd:TIME:SETup? query returns the currently specified setup time.

Return Format

```
<time><NL>

<time> ::= floating-point number in NR3 format
```

See Also

- "Introduction to :TRIGger Commands" on page 1077

:TRIGger:TRANsition Commands

The :TRIGger:TRANsition commands set the rise/fall time trigger options.

Table 141 :TRIGger:TRANsition Commands Summary

Command	Query	Options and Query Returns
:TRIGger:TRANsition:QUALifier <qualifier> (see page 1131)	:TRIGger:TRANsition:QUALifier? (see page 1131)	<qualifier> ::= {GREaterthan | LESSthan}
:TRIGger:TRANsition:SLOPe <slope> (see page 1132)	:TRIGger:TRANsition:SLOPe? (see page 1132)	<slope> ::= {NEGative | POSitive}
:TRIGger:TRANsition:SOURce <source> (see page 1133)	:TRIGger:TRANsition:SOURce? (see page 1133)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:TRANsition:TIME <time>[suffix] (see page 1134)	:TRIGger:TRANsition:TIME? (see page 1134)	<time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}

:TRIGger:TRANsition:QUALifier

(see page 1292)

Command Syntax

```
:TRIGger:TRANsition:QUALifier <qualifier>

<qualifier> ::= {GREaterthan | LESSthan}
```

The :TRIGger:TRANsition:QUALifier command specifies whether you are looking for rise/fall times greater than or less than a certain time value. The time value is set using the :TRIGger:TRANsition:TIME command.

Query Syntax

```
:TRIGger:TRANsition:QUALifier?
```

The :TRIGger:TRANsition:QUALifier? query returns the current rise/fall time trigger qualifier setting.

Return Format

```
<qualifier><NL>

<qualifier> ::= {GRE | LESS}
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:TRANsition:TIME" on page 1134
- ":TRIGger:MODE" on page 1091

:TRIGger:TRANsition:SLOPe

(see page 1292)

Command Syntax

```
:TRIGger:TRANsition:SLOPe <slope>

<slope> ::= {NEGative | POSitive}
```

The :TRIGger:TRANsition:SLOPe command specifies a POSitive rising edge or a NEGative falling edge.

Query Syntax

```
:TRIGger:TRANsition:SLOPe?
```

The :TRIGger:TRANsition:SLOPe? query returns the current rise/fall time trigger slope setting.

Return Format

```
<slope><NL>

<slope> ::= {NEG | POS}
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:MODE" on page 1091
- ":TRIGger:TRANsition:SOURce" on page 1133

:TRIGger:TRANsition:SOURce

(see page 1292)

Command Syntax

```
:TRIGger:TRANsition:SOURce <source>

<source> ::= CHANnel<n>

<n> ::= 1 to (# analog channels) in NR1 format
```

The :TRIGger:TRANsition:SOURce command selects the channel used to produce the trigger.

Query Syntax

```
:TRIGger:TRANsition:SOURce?
```

The :TRIGger:TRANsition:SOURce? query returns the current transition trigger source.

Return Format

```
<source><NL>

<source> ::= CHAN<n>
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:MODE" on page 1091
- ":TRIGger:TRANsition:SLOPe" on page 1132

:TRIGger:TRANsition:TIME

(see page 1292)

Command Syntax

```
:TRIGger:TRANsition:TIME <time>[suffix]

<time> ::= floating-point number in NR3 format

[suffix] ::= {s | ms | us | ns | ps}
```

The :TRIGger:TRANsition:TIME command sets the time value for rise/fall time triggers. You also use the :TRIGger:TRANsition:QUALifier command to specify whether you are triggering on times greater than or less than this time value.

Query Syntax

```
:TRIGger:TRANsition:TIME?
```

The :TRIGger:TRANsition:TIME? query returns the current rise/fall time trigger time value.

Return Format

```
<time><NL>

<time> ::= floating-point number in NR3 format
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":TRIGger:TRANsition:QUALifier" on page 1131

:TRIGger:ZONE Commands

Table 142 :TRIGger:ZONE Commands Summary

Command	Query	Options and Query Returns
:TRIGger:ZONE<z>:LOGic {AND | OR} (see page 1136)	:TRIGger:ZONE<z>:LOGic? (see page 1140)	{AND | OR} <z> ::= 2-4 in NR1 format
:TRIGger:ZONE<z>:MODE <mode> (see page 1137)	:TRIGger:ZONE<z>:MODE? (see page 1137)	<mode> ::= {INTersect | NOTintersect} <z> ::= 1-4 in NR1 format
:TRIGger:ZONE<z>:PLACement <width>, <height>, <x_center>, <y_center> (see page 1138)	:TRIGger:ZONE<z>:PLACement? (see page 1138)	<width> ::= width of zone in seconds <height> ::= height of zone in volts <x_center> ::= center of zone in seconds <y_center> ::= center of zone in volts <z> ::= 1-4 in NR1 format
:TRIGger:ZONE<z>:SOURce <source> (see page 1139)	:TRIGger:ZONE<z>:SOURce? (see page 1139)	<source> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format <z> ::= 1-4 in NR1 format
:TRIGger:ZONE<z>:STATe {{0 | OFF} | {1 | ON}} (see page 1140)	:TRIGger:ZONE<z>:STATe? (see page 1140)	{0 | 1} <z> ::= 1-4 in NR1 format
n/a	:TRIGger:ZONE<z>:VALidity? (see page 1141)	<value> ::= {VALid | INValid | OSCReen} <z> ::= 1-4 in NR1 format

:TRIGger:ZONE<z>:LOGic

 (see page 1292)

Command Syntax

```
:TRIGger:ZONE<z>:LOGic {AND | OR}

<z> ::= 2-4 in NR1 format
```

For Zones 2, 3, or 4, the :TRIGger:ZONE<z>:LOGic command sets the logic to be used when combining with the previous Zone.

The overall Zone Logic is displayed in the grid and in the Zone Trigger dialog box. AND combinations occur before OR combinations.

Query Syntax

```
:TRIGger:ZONE<n>:LOGic?
```

The :TRIGger:ZONE<n>:LOGic? query returns the logic to be used when combining with the previous Zone.

Return Format

```
<logic><NL>

<logic> ::= {AND | OR}
```

See Also

- ":TRIGger:ZONE<z>:STATe" on page 1140

:TRIGger:ZONE<z>:MODE

 (see page 1292)

Command Syntax

```
:TRIGger:ZONE<z>:MODE <mode>

<mode> ::= {INTersect | NOTintersect}

<z> ::= 1-4 in NR1 format
```

The :TRIGger:ZONE<z>:MODE command sets the zone qualifying condition for the Zone as either "Must Intersect" or "Must Not Intersect".

Query Syntax

```
:TRIGger:ZONE<z>:MODE?
```

The :TRIGger:ZONE<z>:MODE? query returns the zone qualifying condition for the Zone.

Return Format

```
<mode><NL>

<mode> ::= {INT | NOT}
```

See Also

- ":TRIGger:ZONE<z>:STATe" on page 1140
- ":TRIGger:ZONE<z>:PLACement" on page 1138
- ":TRIGger:ZONE<z>:VALidity?" on page 1141

:TRIGger:ZONE<z>:PLACement

 (see page 1292)

Command Syntax

```
:TRIGger:ZONE<z>:PLACement <width>, <height>, <x_center>, <y_center>

<width> ::= width of zone in seconds

<height> ::= height of zone in volts

<x_center> ::= center of zone in seconds

<y_center> ::= center of zone in volts

<z> ::= 1-4 in NR1 format
```

The :TRIGger:ZONE<z>:PLACement command sets the size and location of a Zone.

No error is returned if the zone is placed off-screen, or if the zones overlap such that the Zone becomes invalid. The :TRIGger:ZONE<z>:VALidity? query is used to retrieve this information.

Query Syntax

```
:TRIGger:ZONE<z>:PLACement?
```

The :TRIGger:ZONE<z>:PLACement? query returns the size and location of the Zone.

Return Format

```
<opt><NL>

<opt> ::= <width>, <height>, <x_center>, <y_center>
```

See Also

- ":TRIGger:ZONE<z>:STATe" on page 1140
- ":TRIGger:ZONE<z>:MODE" on page 1137
- ":TRIGger:ZONE<z>:VALidity?" on page 1141

:TRIGger:ZONE<z>:SOURce

N (see page 1292)

Command Syntax

```
:TRIGger:ZONE<z>:SOURce <source>

<source> ::= {CHANnel<n>}

<n> ::= 1 to (# analog channels) in NR1 format

<z> ::= 1-4 in NR1 format
```

The :TRIGger:ZONE<z>:SOURce command sets the analog source channel for a particular Zone.

Query Syntax

```
:TRIGger:ZONE<z>:SOURce?
```

The :TRIGger:ZONE<z>:SOURce? query returns the analog source channel specified for a particular Zone.

Return Format

```
<source><NL>

<source> ::= {CHAN<n>}
```

See Also

- ":TRIGger:ZONE<z>:MODE" on page 1137
- ":TRIGger:ZONE<z>:PLACement" on page 1138
- ":TRIGger:ZONE<z>:STATe" on page 1140
- ":TRIGger:ZONE<z>:VALidity?" on page 1141

:TRIGger:ZONE<z>:STATe

(see page 1292)

Command Syntax

```
:TRIGger:ZONE<z>:STATe <on_off>

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

<z> ::= 1-4 in NR1 format
```

The :TRIGger:ZONE<n>:STATe command specifies whether a Zone is ON or OFF.

When a zone's state is on, that zone is actively being used to qualify the trigger if it is not invalid or off-screen (see ":TRIGger:ZONE<z>:VALidity?" on page 1141).

Query Syntax

```
:TRIGger:ZONE<z>:STATe?
```

The :TRIGger:ZONE<z>:STATe? query returns whether a Zone is ON or OFF.

Return Format

```
<on_off><NL>

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

See Also

- ":TRIGger:ZONE<z>:VALidity?" on page 1141

:TRIGger:ZONE<z>:VALidity?

N (see page 1292)

Query Syntax

```
:TRIGger:ZONE<z>:VALidity?

<z> ::= 1-4 in NR1 format
```

The :TRIGger:ZONE<z>:VALidity? query returns the validity of the specified Zone.

- INValid is returned when a Zone overlaps and has opposing qualifying conditions (modes). Zone 1 can never be invalid.
- OSCReen (off-screen) is returned when the associated zone is off-screen, and thus not being used to qualify the trigger.
- A zone is valid when it is neither invalid nor off-screen.

The validity of a zone is not affected by the zone's state. For example, a zone can be valid and off. You cannot directly set the validity of a zone.

Return Format

```
<validity><NL>

<validity> ::= {VALid | INValid | OSCReen}
```

See Also

- ":TRIGger:ZONE<z>:STATe" on page 1140
- ":TRIGger:ZONE<z>:MODE" on page 1137
- ":TRIGger:ZONE<z>:PLACement" on page 1138

36 :WAVeform Commands

Provide access to waveform data. See "Introduction to :WAVeform Commands" on page 1145.

Table 143 :WAVeform Commands Summary

Command	Query	Options and Query Returns
:WAVeform:BYTeorder <value> (see page 1151)	:WAVeform:BYTeorder? (see page 1151)	<value> ::= {LSBFirst | MSBFirst}
n/a	:WAVeform:COUNt? (see page 1152)	<count> ::= an integer from 1 to 65536 in NR1 format
n/a	:WAVeform:DATA? (see page 1153)	<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> (see page 1155)	:WAVeform:FORMat? (see page 1155)	<value> ::= {WORD | BYTE | ASCII}
:WAVeform:POINts <wfm_points> (see page 1156)	:WAVeform:POINts? (see page 1156)	<wfm_points> ::= <#_points> <#_points> ::= an integer in NR1 format
:WAVeform:POINts:MODE <points_mode> (see page 1158)	:WAVeform:POINts:MODE? (see page 1159)	<points_mode> ::= {NORMal | MAXimum | RAW}

Table 143 :WAVeform Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:WAVeform:PREamble? (see page 1160)	<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 • 3 for AVERage type <count> ::= Average count, or 1 if PEAK detect type or NORMal; an integer in NR1 format
:WAVeform:SEGMented:ALL {{0 | OFF} | {1 | ON}} (see page 1163)	:WAVeform:SEGMented:ALL? (see page 1163)	<setting> ::= {0 | 1}
n/a	:WAVeform:SEGMented:COUNt? (see page 1164)	<count> ::= an integer from 2 to 1000 in NR1 format
n/a	:WAVeform:SEGMented:TTAG? (see page 1165)	<time_tag> ::= in NR3 format
n/a	:WAVeform:SEGMented:XLISt? <xlist_type> (see page 1166)	<xlist_type> ::= {RELXorigin | ABSXorigin | TTAG} <return_value> ::= X-info for all segments
:WAVeform:SOURce <source> (see page 1167)	:WAVeform:SOURce? (see page 1167)	<source> ::= {CHANnel<n> | POD{1 | 2} | BUS{1 | 2} | FUNCtion<m> | MATH<m> | FFT | SBUS} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format
:WAVeform:SOURce:SUBSource <subsource> (see page 1171)	:WAVeform:SOURce:SUBSource? (see page 1171)	<subsource> ::= {{SUB0 | RX | MOSI} | {SUB1 | TX | MISO}}

Table 143 :WAVeform Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:WAVeform:TYPE? (see page 1172)	<return_mode> ::= {NORM | PEAK | AVER}
:WAVeform:UNSigned {{0 | OFF} | {1 | ON}} (see page 1173)	:WAVeform:UNSigned? (see page 1173)	{0 | 1}
:WAVeform:VIEW <view> (see page 1174)	:WAVeform:VIEW? (see page 1174)	<view> ::= {MAIN | ALL}
n/a	:WAVeform:XINCrement? (see page 1175)	<return_value> ::= x-increment in the current preamble in NR3 format
n/a	:WAVeform:XORigin? (see page 1176)	<return_value> ::= x-origin value in the current preamble in NR3 format
n/a	:WAVeform:XREFerence? (see page 1177)	<return_value> ::= 0 (x-reference value in the current preamble in NR1 format)
n/a	:WAVeform:YINCrement? (see page 1178)	<return_value> ::= y-increment value in the current preamble in NR3 format
n/a	:WAVeform:YORigin? (see page 1179)	<return_value> ::= y-origin in the current preamble in NR3 format
n/a	:WAVeform:YREFerence? (see page 1180)	<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 (see page 1153) and :WAVeform:PREamble (see page 1160). 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 four types of waveform acquisitions that can be selected for analog channels with the :ACQuire:TYPE command (see page 247): NORMal, AVERage, and PEAK. Digital channels are always acquired using NORMal. When the data is acquired using the :DIGitize command (see page 218) or :RUN command (see page 221), 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 the programming interface 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 (see page 1153) 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[:ANALog]? (see page 238).

Helpful Hints:

The number of points transferred to the computer is controlled using the :WAVeform:POINts command (see page 1156). 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 the programming interface 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 (see page 1156). 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 (see page 234). 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 (see page 1156). 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 (see page 1156). 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 (see page 247), the value returned by the :WAVeform:XINCrement query (see page 1175) should be doubled to find the time difference between the min-max pairs.

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 (see page 1155), 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 (see page 247), 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" on page 1155). 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 (see page 1173) 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" on page 1155) 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 (see page 1151) and :WAVeform:UNSigned (see page 1173) have no effect when the format is ASCii.

Data Format for Transfer - WORD format

WORD format (see ":WAVeform:FORMat" on page 1155) 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 (see page 1156). 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 (see page 1151) 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" on page 1155) 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 programming interface 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 (see page 1151) has no effect when the data format is BYTE.

Digital Channel Data

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 (see page 1173) 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 (see page 1155) is WORD, every other data byte will be 0. The setting of :WAVeform:BYTeorder (see page 1151) 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 (see page 249) 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 BYTE;POIN +32000;SOUR CHAN1;
SOUR:SUBS SUB0
```

:WAVeform:BYTeorder

(see page 1292)

Command Syntax

```
:WAVeform:BYTeorder <value>

<value> ::= {LSBFirst | MSBFirst}
```

The :WAVeform:BYTeorder command sets the output sequence of the WORD data.

- MSBFirst – sets the most significant byte to be transmitted first.
- 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 MSBFirst.

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" on page 1145
- ":WAVeform:DATA?" on page 1153
- ":WAVeform:FORMat" on page 1155
- ":WAVeform:PREamble?" on page 1160

Example Code

- "Example Code" on page 1168
- "Example Code" on page 1161

:WAVeform:COUNt?

 (see page 1292)

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" on page 1145
- ":ACQuire:COUNt" on page 234
- ":ACQuire:TYPE" on page 247

:WAVeform:DATA?

(see page 1292)

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.

In BYTE or WORD waveform formats, these data values have special meaning:

- 0x00 or 0x0000 – Hole. Holes are locations where data has not yet been acquired.

  Another situation where there can be zeros in the data, incorrectly, is when programming over telnet port 5024. Port 5024 provides a command prompt and is intended for ASCII transfers. Use telnet port 5025 instead.
- 0x01 or 0x0100 – Clipped low. These are locations where the waveform is clipped at the bottom of the oscilloscope display.
- 0xFF or 0xFF00 – Clipped high. These are locations where the waveform is clipped at the top of the oscilloscope display.

Return Format

```
<binary block data><NL>
```

See Also

- For a more detailed description of the data returned for different acquisition types, see: "Introduction to :WAVeform Commands" on page 1145
- ":WAVeform:UNSigned" on page 1173
- ":WAVeform:BYTeorder" on page 1151
- ":WAVeform:FORMat" on page 1155
- ":WAVeform:POINts" on page 1156
- ":WAVeform:PREamble?" on page 1160
- ":WAVeform:SOURce" on page 1167
- ":WAVeform:TYPE?" on page 1172

Example Code

```
' QUERY_WAVE_DATA - Outputs waveform data that is stored in a buffer.

' Query the oscilloscope for the waveform data.
myScope.WriteString ":WAV: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
```

See complete example programs at: Chapter 44, “Programming Examples,” starting on page 1301

:WAVeform:FORMat

(see page 1292)

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 (SBUS1 or SBUS2), 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" on page 1145
- ":WAVeform:BYTeorder" on page 1151
- ":WAVeform:SOURce" on page 1167
- ":WAVeform:DATA?" on page 1153
- ":WAVeform:PREamble?" on page 1160

Example Code

- "Example Code" on page 1168

:WAVeform:POINts

(see page 1292)

Command Syntax

```
:WAVeform:POINts <#_points>

<#_points> ::= an integer in NR1 format
```

The :WAVeform:POINts command sets the desired 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, the records that can be transferred depend on the waveform points mode. The maximum number of points returned for math (function) waveforms is determined by the NORMal waveform points mode. See the :WAVeform:POINts:MODE command (see page 1158) for more information.

To get the maximum number of points possible, you can set the waveform points mode to MAXimum and ask for the maximum memory depth of the oscilloscope.

Only data visible on the display will be returned.

When the :WAVeform:SOURce is the serial decode bus (SBUS1 or SBUS2), 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 (see page 1158) for more information).

In any waveform points mode, the oscilloscope returns as close to the requested number of points as possible, but the actual number of points available depends on oscilloscope settings.

When the :WAVeform:SOURce is the serial decode bus (SBUS1 or SBUS2), this query returns the number of messages that were decoded.

Return Format

```
<#_points><NL>

<#_points> ::= an integer number of waveform points
```

See Also

- "Introduction to :WAVeform Commands" on page 1145
- ":ACQuire:POINts[:ANALog]" on page 238
- ":WAVeform:DATA?" on page 1153
- ":WAVeform:SOURce" on page 1167
- ":WAVeform:VIEW" on page 1174
- ":WAVeform:PREamble?" on page 1160
- ":WAVeform:POINts:MODE" on page 1158

Example Code

```
' WAVE_POINTS - Specifies the number of points to be transferred
' using the ":WAVeform:DATA?" query.
myScope.WriteString ":WAVeform:POINTS 1000"
```

See complete example programs at: Chapter 44, “Programming Examples,” starting on page 1301

:WAVeform:POINts:MODE

N (see page 1292)

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 retrieved from the analog or digital sources.
- The second is referred to as the *analysis record* and defaults to a 64K-point representation of the raw acquisition record. The analysis record length can be increased at the expense of waveform update rate (see :SYSTem:PRECision:LENGth). The analysis record can be retrieved from any source.

If the <points_mode> is NORMal the analysis record is retrieved.

If the <points_mode> is RAW, the raw acquisition record is used. Under some conditions, 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, the analysis 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 analysis record and raw acquisition records.

NOTE

If the :WAVeform:SOURce is not an analog or digital source, the only valid parameters for WAVeform:POINts:MODE is NORMal or MAXimum.

Considerations for MAXimum or RAW data retrieval

- The instrument must be stopped (see the :STOP command (see page 225) or the :DIGitize command (see page 218) in the root subsystem) in order to return more than the *analysis record*.
- :TIMebase:MODE must be set to MAIN.
- :ACQuire:TYPE must be set to NORMal or AVERage.
- MAXimum or RAW will allow up to the maximum memory depth number of 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" on page 1145
- ":WAVeform:DATA?" on page 1153
- ":ACQuire:POINts[:ANALog]" on page 238
- ":SYSTem:PRECision:LENGth" on page 1043
- ":WAVeform:VIEW" on page 1174
- ":WAVeform:PREamble?" on page 1160
- ":WAVeform:POINts" on page 1156
- ":TIMebase:MODE" on page 1070
- ":ACQuire:TYPE" on page 247
- ":ACQuire:COUNt" on page 234

:WAVeform:PREamble?

(see page 1292)

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

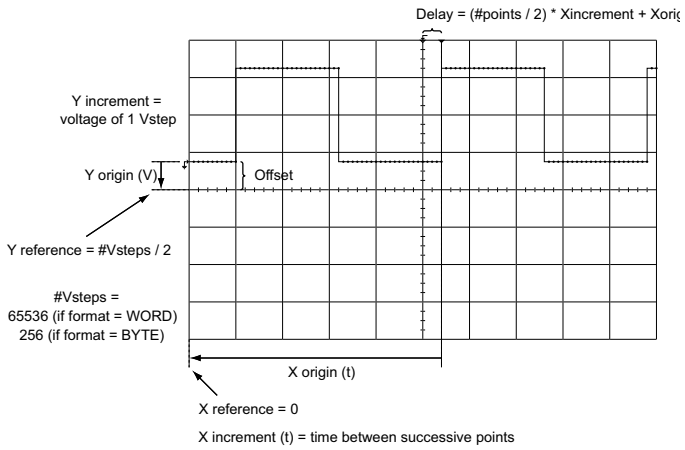
```
<preamble_block><NL>

<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, 4 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).
```



See Also

- "Introduction to :WAVeform Commands" on page 1145
- ":ACQuire:COUNt" on page 234
- ":ACQuire:POINts[:ANALog]" on page 238

- ":ACQuire:TYPE" on page 247
- ":DIGitize" on page 218
- ":WAVeform:COUNt?" on page 1152
- ":WAVeform:DATA?" on page 1153
- ":WAVeform:FORMat" on page 1155
- ":WAVeform:POINts" on page 1156
- ":WAVeform:TYPE?" on page 1172
- ":WAVeform:XINCrement?" on page 1175
- ":WAVeform:XORigin?" on page 1176
- ":WAVeform:XREFerence?" on page 1177
- ":WAVeform:YINCrement?" on page 1178
- ":WAVeform:YORigin?" on page 1179
- ":WAVeform:YREFerence?" on page 1180

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, 4 = ASCII.
'    TYPE          : int16 - 0 = NORM, 1 = PEAK, 2 = AVER.
'    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 diff 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)
```

See complete example programs at: Chapter 44, “Programming Examples,” starting on page 1301

:WAVeform:SEGMented:ALL

N (see page 1292)

Command Syntax `:WAVeform:SEGMented:ALL {{0 | OFF} | {1 | ON}}`

The :WAVeform:SEGMented:ALL command enables or disables the "waveform data for all segments" setting:

- ON – The :WAVeform:DATA? query returns data for all segments.

  ON is available only when the :WAVeform:SOURce is an analog input channel.

  When ON, the :WAVeform:DATA? query returns data for the number of segments multiplied by the number of points in a segment (which is the points value returned by the :WAVeform:POINts? or :WAVeform:PREamble? queries).
- OFF – The :WAVeform:DATA? query returns data for the current segment.

The ability to get waveform data for all segments is faster and more convenient than iterating over multiple segments and retrieving each one separately.

The performance improvement comes primarily when raw acquisition record data is being retrieved (:WAVeform:POINts:MODE RAW) instead of the analysis record data (:WAVeform:POINts:MODE NORMal).

One corner case where you may want to retreive the analysis record data is when the acquired data for a segment is less than the screen resolution and the analysis record is interpolated.

Most often though, the acquired data for a segment is enough to fill the screen, and less than the maximum analysis record size, so the segmented raw acquisition data and the analysis record data are the same.

If the number of segments is small, the raw acquisition segment size is greater than the analysis record size.

Query Syntax `:WAVeform:SEGMented:ALL?`

The :WAVeform:SEGMented:ALL? query returns the "waveform data for all segments" setting.

Return Format `<setting><NL>`

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

See Also

- ":WAVeform:DATA?" on page 1153
- ":WAVeform:SEGMented:XLISt?" on page 1166
- ":WAVeform:POINts:MODE" on page 1158
- ":WAVeform:POINts" on page 1156
- ":WAVeform:PREamble?" on page 1160

:WAVeform:SEGMented:COUNt?

(see page 1292)

Query Syntax

```
:WAVeform:SEGMented:COUNt?
```

The :WAVeform:SEGMented:COUNt query returns the number of memory segments in the acquired data. You can use the :WAVeform:SEGMented:COUNt? query while segments are being acquired (although :DIGitize blocks subsequent queries until the full segmented acquisition is complete).

The segmented memory acquisition mode is enabled with the :ACQuire:MODE command. The number of segments to acquire is set using the :ACQuire:SEGMented:COUNt command, and data is acquired using the :DIGitize, :SINGle, or :RUN commands.

Return Format

```
<count> ::= an integer from 2 to 1000 in NR1 format (count set by
            :ACQuire:SEGMented:COUNt).
```

See Also

- ":ACQuire:MODE" on page 237
- ":ACQuire:SEGMented:COUNt" on page 241
- ":DIGitize" on page 218
- ":SINGle" on page 223
- ":RUN" on page 221
- "Introduction to :WAVeform Commands" on page 1145

Example Code

- "Example Code" on page 242

:WAVeform:SEGMented:TTAG?

 (see page 1292)

Query Syntax `:WAVeform:SEGMented:TTAG?`

The :WAVeform:SEGMented:TTAG? query returns the time tag of the currently selected segmented memory index. The index is selected using the :ACQuire:SEGMented:INDex command.

Return Format `<time_tag> ::= in NR3 format`

See Also

- ":ACQuire:SEGMented:INDex" on page 242
- "Introduction to :WAVeform Commands" on page 1145

Example Code

- "Example Code" on page 242

:WAVeform:SEGMented:XLISt?

(see page 1292)

Query Syntax

```
:WAVeform:SEGMented:XLISt? <xlist_type>

<xlist_type> ::= {RELXorigin | ABSXorigin | TTAG}
```

The :WAVeform:SEGMented:XLISt? query returns the X (time) information for all segments at once. The <xlist_type> option specifies the type of information that is returned:

- RELXorigin — The relative X-origin for each segment (the value returned by the :WAVeform:XORigin? query) is returned.
- TTAG — The time tag for each segment (the value returned by the :WAVeform:SEGMented:TTAG? query) will be returned.
- ABSXorigin — The sum of the values of the RELXorigin and TTAG types is returned for each segment.

This command is useful when getting the waveform data for all segments at once (see :WAVeform:SEGMented:ALL).

Return Format

```
<return_value><NL>

<return_value> ::= binary block data in IEEE 488.2 # format, contains
   comma-separated string with X-info for all segments
```

See Also

- ":WAVeform:XORigin?" on page 1176
- ":WAVeform:SEGMented:TTAG?" on page 1165
- ":WAVeform:SEGMented:ALL" on page 1163

:WAVeform:SOURce

(see page 1292)

Command Syntax

```
:WAVeform:SOURce <source>

<source> ::= {CHANnel<n> | POD{1 | 2} | BUS{1 | 2} | FUNCtion<m>
             | MATH<m> | FFT | WMEMory<r> | SBUS{1 | 2}}

<n> ::= 1 to (# analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :WAVeform:SOURce command selects the analog channel, function, digital pod, digital bus, reference waveform, 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.

When the :WAVeform:SOURce is the serial decode bus (SBUS1 or SBUS2), ASCii is the only waveform format allowed, and the :WAVeform:DATA? query returns a string with timestamps and associated bus decode information.

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" on page 1155).

  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 hexadecimal bus values, for example: 0x1938,0xff38,...

Query Syntax

```
:WAVeform:SOURce?
```

The :WAVeform:SOURce? query returns the currently selected source for the WAVeform commands.

NOTE MATH<m> is an alias for FUNCtion<m>. The :WAVeform:SOURce? query returns FUNC<m> if the source is FUNCtion<m> or MATH<m>.

Return Format

```
<source><NL>

<source> ::= {CHAN<n> | POD{1 | 2} | BUS{1 | 2} | FUNC<m>
             | WMEM<r> | SBUS{1 | 2}}

<n> ::= 1 to (# analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

See Also

- "Introduction to :WAVeform Commands" on page 1145
- ":DIGitize" on page 218
- ":WAVeform:FORMat" on page 1155
- ":WAVeform:BYTeorder" on page 1151
- ":WAVeform:DATA?" on page 1153
- ":WAVeform:PREamble?" on page 1160

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

' Data in range 0 to 65535.
myScope.WriteString ":WAVeform:FORMAT WORD"
lngVSteps = 65536
intBytesPerData = 2

' Data in range 0 to 255.
'myScope.WriteString ":WAVeform:FORMat BYTE"
'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, 4 = 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 diff 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.

' Query the oscilloscope for the waveform data.
myScope.WriteString ":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

' Unsigned integer bytes.
varQueryResult = myScope.ReadIEEEBlock(BinaryType_UI1)

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
```

See complete example programs at: Chapter 44, “Programming Examples,” starting on page 1301

:WAVeform:SOURce:SUBSource

(see page 1292)

Command Syntax

```
:WAVeform:SOURce:SUBSource <subsource>

<subsource> ::= {{SUB0 | RX | MOSI | FAST}
                 | {SUB1 | TX | MISO | SLOW}}
```

If the :WAVeform:SOURce is SBUS<n> (serial decode), more than one data set may be available, and this command lets you choose from the available data sets.

When using UART serial decode, this option lets you get "TX" data. (TX is an alias for SUB1.) The default, SUB0, specifies "RX" data. (RX is an alias for SUB0.)

When using SPI serial decode, this option lets you get "MISO" data. (MISO is an alias for SUB1.) The default, SUB0, specifies "MOSI" data. (MOSI is an alias for SUB0.)

If the :WAVeform:SOURce is not SBUS, or the :SBUS<n>:MODE is not UART or SPI, the only valid subsource is SUB0.

Query Syntax

```
:WAVeform:SOURce:SUBSource?
```

The :WAVeform:SOURce:SUBSource? query returns the current waveform subsource setting.

Return Format

```
<subsource><NL>

<subsource> ::= {SUB0 | SUB1}
```

See Also

- "Introduction to :WAVeform Commands" on page 1145
- ":WAVeform:SOURce" on page 1167

:WAVeform:TYPE?

(see page 1292)

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}`

NOTE If the :WAVeform:SOURce is POD1, POD2, or SBUS1, SBUS2, the type is always NORM.

See Also

- "Introduction to :WAVeform Commands" on page 1145
- ":ACQuire:TYPE" on page 247
- ":WAVeform:DATA?" on page 1153
- ":WAVeform:PREamble?" on page 1160
- ":WAVeform:SOURce" on page 1167

:WAVeform:UNSigned

 (see page 1292)

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, POD2, BUS1, or BUS2, 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" on page 1145
- ":WAVeform:SOURce" on page 1167

:WAVeform:VIEW

(see page 1292)

Command Syntax

```
:WAVeform:VIEW <view>

<view> ::= {MAIN | ALL}
```

The :WAVeform:VIEW command sets the view setting associated with the currently selected waveform:

- MAIN — This view specifies the data you see in the oscilloscope's main waveform display area.
- ALL — Available only when Digitizer mode is on (see :ACQuire:DIGitizer), this view specifies all the captured data, which may extend beyond the edges of the oscilloscope's main waveform display area depending on the settings for sample rate, memory depth, and horizontal time/div.

Query Syntax

```
:WAVeform:VIEW?
```

The :WAVeform:VIEW? query returns the view setting associated with the currently selected waveform.

Return Format

```
<view><NL>

<view> ::= {MAIN | ALL}
```

See Also

- "Introduction to :WAVeform Commands" on page 1145
- ":WAVeform:POINts" on page 1156
- ":ACQuire:DIGitizer" on page 235

:WAVeform:XINCrement?

(see page 1292)

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" on page 1145
- ":WAVeform:PREamble?" on page 1160

Example Code

- "Example Code" on page 1161

:WAVeform:XORigin?

(see page 1292)

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" on page 1145
- ":WAVeform:PREamble?" on page 1160
- ":WAVeform:XREFerence?" on page 1177

Example Code

- "Example Code" on page 1161

:WAVeform:XREFerence?

 (see page 1292)

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" on page 1145
- ":WAVeform:PREamble?" on page 1160
- ":WAVeform:XORigin?" on page 1176

Example Code

- "Example Code" on page 1161

:WAVeform:YINCrement?

(see page 1292)

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" on page 1145
- ":WAVeform:PREamble?" on page 1160

Example Code

- "Example Code" on page 1161

:WAVeform:YORigin?

(see page 1292)

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" on page 1145
- ":WAVeform:PREamble?" on page 1160
- ":WAVeform:YREFerence?" on page 1180

Example Code

- "Example Code" on page 1161

:WAVeform:YREFerence?

(see page 1292)

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" on page 1145
- ":WAVeform:PREamble?" on page 1160
- ":WAVeform:YORigin?" on page 1179

Example Code

- "Example Code" on page 1161

37 :WGEN<w> Commands

When the built-in waveform generator is licensed (WAVEGEN license), you can use it to output sine, square, ramp, pulse, DC, noise, sine cardinal, exponential rise, exponential fall, cardiac, and gaussian pulse waveforms. The :WGEN<w> commands are used to select the waveform function and parameters. See "Introduction to :WGEN<w> Commands" on page 1184.

Table 144 :WGEN<w> Commands Summary

Command	Query	Options and Query Returns
:WGEN<w>:ARBitrary:BYTeorder <order> (see page 1186)	:WGEN<w>:ARBitrary:BYTeorder? (see page 1186)	<order> ::= {MSBFirst | LSBFirst} <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:ARBitrary:DATA {<binary> | <value>, <value> ...} (see page 1187)	n/a	<binary> ::= floating point values between -1.0 to +1.0 in IEEE 488.2 binary block format <value> ::= floating point values between -1.0 to +1.0 in comma-separated format <w> ::= 1 to (# WaveGen outputs) in NR1 format
n/a	:WGEN<w>:ARBitrary:DATA:ATTRibute:POINts? (see page 1190)	<points> ::= number of points in NR1 format <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:ARBitrary:DATA:CLEar (see page 1191)	n/a	<w> ::= 1 to (# WaveGen outputs) in NR1 format

Table 144 :WGEN<w> Commands Summary (continued)

Command	Query	Options and Query Returns
:WGEN<w>:ARBitrary:DATA:DAC {<binary> | <value>, <value> ...} (see page 1192)	n/a	<binary> ::= decimal 16-bit integer values between -512 to +511 in IEEE 488.2 binary block format <value> ::= decimal integer values between -512 to +511 in comma-separated NR1 format <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:ARBitrary:INTerpolate {{0 | OFF} | {1 | ON}} (see page 1193)	:WGEN<w>:ARBitrary:INTerpolate? (see page 1193)	{0 | 1} <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:ARBitrary:STORe <source> (see page 1194)	n/a	<source> ::= {CHANnel<n> | WMEMory<r> | FUNCtion<m> | FFT | MATH<m>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:DCMode <mode> (see page 1195)	:WGEN<w>:DCMode? (see page 1195)	<mode> ::= {PRECise | WIDerange}
:WGEN<w>:FREQuency <frequency> (see page 1196)	:WGEN<w>:FREQuency? (see page 1196)	<frequency> ::= frequency in Hz in NR3 format <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:FUNCtion <signal> (see page 1197)	:WGEN<w>:FUNCtion? (see page 1201)	<signal> ::= {SINusoid | SQUare | RAMP | PULSe | NOISe | DC | SINC | EXPRise | EXPFall | CARDiac | GAUSsian | ARBitrary} <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:FUNCtion:PULSe:WIDTh <width> (see page 1202)	:WGEN<w>:FUNCtion:PULSe:WIDTh? (see page 1202)	<width> ::= pulse width in seconds in NR3 format <w> ::= 1 to (# WaveGen outputs) in NR1 format

Table 144 :WGEN<w> Commands Summary (continued)

Command	Query	Options and Query Returns
:WGEN<w>:FUNCtion:SQUare:DCYCle <percent> (see page 1204)	:WGEN<w>:FUNCtion:SQUare:DCYCle? (see page 1204)	<percent> ::= duty cycle percentage from 20% to 80% in NR1 format <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:MODulation:AM:DEPTh <percent> (see page 1205)	:WGEN<w>:MODulation:AM:DEPTh? (see page 1205)	<percent> ::= AM depth percentage from 0% to 100% in NR1 format <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:AM:FREQuency <frequency> (see page 1206)	:WGEN<w>:MODulation:AM:FREQuency? (see page 1206)	<frequency> ::= modulating waveform frequency in Hz in NR3 format <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:FM:DEViation <frequency> (see page 1207)	:WGEN<w>:MODulation:FM:DEViation? (see page 1207)	<frequency> ::= frequency deviation in Hz in NR3 format <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:FM:FREQuency <frequency> (see page 1208)	:WGEN<w>:MODulation:FM:FREQuency? (see page 1208)	<frequency> ::= modulating waveform frequency in Hz in NR3 format <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:FUNCtion <shape> (see page 1209)	:WGEN<w>:MODulation:FUNCtion? (see page 1209)	<shape> ::= {SINusoid} <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:NOISe <percent> (see page 1210)	:WGEN<w>:MODulation:NOISe? (see page 1210)	<percent> ::= 0 to 100 <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:STATe {{0 | OFF} | {1 | ON}} (see page 1211)	:WGEN<w>:MODulation:STATe? (see page 1211)	{0 | 1} <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:TYPE <type> (see page 1212)	:WGEN<w>:MODulation:TYPE? (see page 1212)	<type> ::= {AM | FM} <w> ::= 1 in NR1 format
:WGEN<w>:OUTPut {{0 | OFF} | {1 | ON}} (see page 1213)	:WGEN<w>:OUTPut? (see page 1213)	{0 | 1} <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:OUTPut:LOAD <impedance> (see page 1214)	:WGEN<w>:OUTPut:LOAD? (see page 1214)	<impedance> ::= {ONEMeg | FIFTy} <w> ::= 1 to (# WaveGen outputs) in NR1 format

Table 144 :WGEN<w> Commands Summary (continued)

Command	Query	Options and Query Returns
:WGEN<w>:OUTPut:POLarity <polarity> (see page 1215)	:WGEN<w>:OUTPut:POLarity? (see page 1215)	<polarity> ::= {NORMal | INVerted} <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:PERiod <period> (see page 1216)	:WGEN<w>:PERiod? (see page 1216)	<period> ::= period in seconds in NR3 format <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:RST (see page 1217)	n/a	<w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:VOLTage <amplitude> (see page 1218)	:WGEN<w>:VOLTage? (see page 1218)	<amplitude> ::= amplitude in volts in NR3 format <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:VOLTage:HIGH <high> (see page 1219)	:WGEN<w>:VOLTage:HIGH? (see page 1219)	<high> ::= high-level voltage in volts, in NR3 format <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:VOLTage:LOW <low> (see page 1220)	:WGEN<w>:VOLTage:LOW? (see page 1220)	<low> ::= low-level voltage in volts, in NR3 format <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:VOLTage:OFFSet <offset> (see page 1221)	:WGEN<w>:VOLTage:OFFSet? (see page 1221)	<offset> ::= offset in volts in NR3 format <w> ::= 1 to (# WaveGen outputs) in NR1 format

Introduction to :WGEN<w> Commands

The :WGEN<w> subsystem provides commands to select the waveform generator function and parameters.

In the :WGEN<w> commands, the <w> can be 1 or 2, and :WGEN is equivalent to :WGEN1

Reporting the Setup

Use :WGEN<w>? to query setup information for the WGEN<w> subsystem.

Return Format

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

```
:WGEN:FUNC SIN;OUTP 0;FREQ +1.00000E+03;VOLT +500.0E-03;
VOLT:OFFS +0.0E+00;:WGEN:OUTP:LOAD ONEM
```

:WGEN<w>:ARBitrary:BYTeorder

(see page 1292)

Command Syntax

```
:WGEN<w>:ARBitrary:BYTeorder <order>

<w> ::= 1 to (# WaveGen outputs) in NR1 format

<order> ::= {MSBFirst | LSBFirst}
```

The :WGEN<w>:ARBitrary:BYTeorder command selects the byte order for binary transfers.

Query Syntax

```
:WGEN<w>:ARBitrary:BYTeorder?
```

The :WGEN<w>:ARBitrary:BYTeorder query returns the current byte order selection.

Return Format

```
<order><NL>

<order> ::= {MSBFirst | LSBFirst}
```

See Also

- ":WGEN<w>:ARBitrary:DATA" on page 1187
- ":WGEN<w>:ARBitrary:DATA:DAC" on page 1192

:WGEN<w>:ARBitrary:DATA

N (see page 1292)

Command Syntax

```
:WGEN<w>:ARBitrary:DATA {<binary> | <value>, <value> ...}

<w> ::= 1 to (# WaveGen outputs) in NR1 format

<binary> ::= floating point values between -1.0 to +1.0
             in IEEE 488.2 binary block format

<value> ::= floating point values between -1.0 to +1.0
            in comma-separated format
```

The :WGEN<w>:ARBitrary:DATA command downloads an arbitrary waveform in floating-point values format.

See Also

- ":WGEN<w>:ARBitrary:DATA:DAC" on page 1192
- ":SAVE:ARBitrary[:STARt]" on page 691
- ":RECall:ARBitrary[:STARt]" on page 678
- "Commands Not Supported in Multiple Program Message Units" on page 1297

Example Code

```
' Waveform generator arbitrary data commands example.
' ----------------------------------------------------------------------

Option Explicit

Public myMgr As VisaComLib.ResourceManager
Public myScope As VisaComLib.FormattedIO488
Public varQueryResult As Variant
Public strQueryResult As String

#If VBA7 Then
    Private Declare PtrSafe Sub Sleep Lib "kernel32" (ByVal dwMillisecon
ds As LongPtr)
    Private Declare PtrSafe Sub CopyMemory Lib "kernel32" Alias "RtlMove
Memory" _
        (dest As Any, source As Any, ByVal bytes As LongPtr)
#Else
    Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Lo
ng)
    Private Declare Sub CopyMemory Lib "kernel32" Alias "RtlMoveMemory"
_
        (dest As Any, source As Any, ByVal bytes As Long)
#End If

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("USB0::0x2A8D::0x4704::Unset::0::INSTR")
  myScope.IO.Clear   ' Clear the interface.
  myScope.IO.Timeout = 20000   ' Set I/O communication timeout.

  ' Turn on arbitrary waveform generator function.
  myScope.WriteString ":WGEN1:OUTPut ON"
  myScope.WriteString ":WGEN1:FUNCtion ARBitrary"
  myScope.WriteString ":WGEN1:FUNCtion?"
  strQueryResult = myScope.ReadString
  Debug.Print "WaveGen1 function: " + strQueryResult

  DefaultArbitraryWaveform

  ' Download comma-separated floating-point values.
  myScope.WriteString ":WGEN1:ARBitrary:DATA 0.0, 0.5, 1.0, 0.5, 0.0, -0
.5, -1.0, -0.5"
  Debug.Print "WaveGen1 CSV floating-point values downloaded."
  Sleep 5000

  DefaultArbitraryWaveform

  ' Download comma-separated 16-bit integer (DAC) values.
  myScope.WriteString ":WGEN1:ARBitrary:DATA:DAC 0, 255, 511, 255, 0, -2
56, -512, -256"
  Debug.Print "WaveGen1 CSV 16-bit integer (DAC) values downloaded."
  Sleep 5000

  ' Set the byte order for binary data.
  myScope.WriteString ":WGEN1:ARBitrary:BYTeorder LSBFirst"
  myScope.WriteString ":WGEN1:ARBitrary:BYTeorder?"
  strQueryResult = myScope.ReadString
  Debug.Print "WaveGen1 byte order for binary data: " + strQueryResult

  DefaultArbitraryWaveform

  ' Download binary floating-point values.
  Dim mySingleArray(8) As Single
  mySingleArray(0) = 0!
  mySingleArray(1) = 0.5!
  mySingleArray(2) = 1!
  mySingleArray(3) = 0.5!
  mySingleArray(4) = 0!
  mySingleArray(5) = -0.5!
  mySingleArray(6) = -1!
  mySingleArray(7) = -0.5!

  Dim myByteArray(32) As Byte
  CopyMemory myByteArray(0), mySingleArray(0), 32 * LenB(myByteArray(0))

  myScope.WriteIEEEBlock ":WGEN1:ARBitrary:DATA", myByteArray, True
  Debug.Print "WaveGen1 binary floating-point values downloaded."
  Sleep 5000

  DefaultArbitraryWaveform

  ' Download binary 16-bit integer (DAC) values.
```

```
  Dim myIntegerArray(8) As Integer
  myIntegerArray(0) = 0
  myIntegerArray(1) = 255
  myIntegerArray(2) = 511
  myIntegerArray(3) = 255
  myIntegerArray(4) = 0
  myIntegerArray(5) = -256
  myIntegerArray(6) = -512
  myIntegerArray(7) = -256

  Dim myByteArray2(16) As Byte
  CopyMemory myByteArray2(0), myIntegerArray(0), 16 * LenB(myByteArray2(
0))

  myScope.WriteIEEEBlock ":WGEN1:ARBitrary:DATA:DAC", myByteArray2, True
  Debug.Print "WaveGen1 binary 16-bit integer (DAC) values downloaded."
  Sleep 5000

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error:" + vbCrLf + Err.Description

End Sub

'
' Initialize WaveGen1 to a known state.
' ----------------------------------------------------------------------
Private Sub DefaultArbitraryWaveform()

  On Error GoTo VisaComError

  ' Load default arbitrary waveform.
  myScope.WriteString ":WGEN1:ARBitrary:DATA:CLEar"
  Debug.Print "WaveGen1 default arbitrary waveform loaded."
  Sleep 5000

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error:" + vbCrLf + Err.Description
  End

End Sub
```

:WGEN<w>:ARBitrary:DATA:ATTRibute:POINts?

(see page 1292)

Query Syntax

```
:WGEN<w>:ARBitrary:DATA:ATTRibute:POINts?

<w> ::= 1 to (# WaveGen outputs) in NR1 format
```

The :WGEN<w>:ARBitrary:DATA:ATTRibute:POINts query returns the number of points used by the current arbitrary waveform.

Return Format

```
<points> ::= number of points in NR1 format
```

See Also

- ":WGEN<w>:ARBitrary:DATA" on page 1187
- ":WGEN<w>:ARBitrary:DATA:DAC" on page 1192
- ":SAVE:ARBitrary[:STARt]" on page 691
- ":RECall:ARBitrary[:STARt]" on page 678

:WGEN<w>:ARBitrary:DATA:CLEar

N (see page 1292)

Command Syntax

```
:WGEN<w>:ARBitrary:DATA:CLEar

<w> ::= 1 to (# WaveGen outputs) in NR1 format
```

The :WGEN<w>:ARBitrary:DATA:CLEar command clears the arbitrary waveform memory and loads it with the default waveform.

See Also

- ":WGEN<w>:ARBitrary:DATA" on page 1187
- ":WGEN<w>:ARBitrary:DATA:DAC" on page 1192
- ":SAVE:ARBitrary[:STARt]" on page 691
- ":RECall:ARBitrary[:STARt]" on page 678

Example Code

- "Example Code" on page 1187

:WGEN<w>:ARBitrary:DATA:DAC

(see page 1292)

Command Syntax

```
:WGEN<w>:ARBitrary:DATA:DAC {<binary> | <value>, <value> ...}

<w> ::= 1 to (# WaveGen outputs) in NR1 format

<binary> ::= decimal 16-bit integer values between -512 to +511
             in IEEE 488.2 binary block format

<value> ::= decimal integer values between -512 to +511
            in comma-separated NR1 format
```

The :WGEN<w>:ARBitrary:DATA:DAC command downloads an arbitrary waveform using 16-bit integer (DAC) values.

See Also

- ":WGEN<w>:ARBitrary:DATA" on page 1187
- ":SAVE:ARBitrary[:STARt]" on page 691
- ":RECall:ARBitrary[:STARt]" on page 678

Example Code

- "Example Code" on page 1187

:WGEN<w>:ARBitrary:INTerpolate

 (see page 1292)

Command Syntax

```
:WGEN<w>:ARBitrary:INTerpolate {{0 | OFF} | {1 | ON}}

<w> ::= 1 to (# WaveGen outputs) in NR1 format
```

The :WGEN<w>:ARBitrary:INTerpolate command enables or disables the Interpolation control.

Interpolation specifies how lines are drawn between arbitrary waveform points:

- When ON, lines are drawn between points in the arbitrary waveform. Voltage levels change linearly between one point and the next.
- When OFF, all line segments in the arbitrary waveform are horizontal. The voltage level of one point remains until the next point.

Query Syntax

```
:WGEN<w>:ARBitrary:INTerpolate?
```

The :WGEN<w>:ARBitrary:INTerpolate query returns the current interpolation setting.

Return Format

```
{0 | 1}
```

See Also

- ":WGEN<w>:ARBitrary:DATA" on page 1187
- ":WGEN<w>:ARBitrary:DATA:DAC" on page 1192
- ":SAVE:ARBitrary[:STARt]" on page 691
- ":RECall:ARBitrary[:STARt]" on page 678

:WGEN<w>:ARBitrary:STORe

(see page 1292)

Command Syntax

```
:WGEN<w>:ARBitrary:STORe <source>

<w> ::= 1 to (# WaveGen outputs) in NR1 format

<source> ::= {CHANnel<n> | WMEMory<r> | FUNCtion<m> | MATH<m> | FFT}

<n> ::= 1 to (# analog channels) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format
```

The :WGEN<w>:ARBitrary:STORe command stores the source's waveform into the arbitrary waveform memory.

See Also

- ":WGEN<w>:ARBitrary:DATA" on page 1187
- ":WGEN<w>:ARBitrary:DATA:DAC" on page 1192
- ":SAVE:ARBitrary[:STARt]" on page 691
- ":RECall:ARBitrary[:STARt]" on page 678

:WGEN<w>:DCMode

N (see page 1292)

Command Syntax

```
:WGEN<w>:DCMode <mode>

<mode> ::= {PRECise | WIDerange}
```

When the DC waveform type is selected (see :WGEN<w>:FUNCtion), the :WGEN<w>:DCMode command specifies the DC output mode:

- PRECise – You can select from a smaller range of output voltages (-1 V to 1 V) with greater accuracy of the specified level.
- WIDerange – You can select from a wider range of output voltages (-8 V to 8 V) with with less accuracy of the specified level.

Query Syntax

```
:WGEN<w>:DCMode?
```

The :WGEN<w>:DCMode? query returns the selected DC output mode selection.

Return Format

```
<mode><NL>

<mode> ::= {PREC | WID}
```

See Also

- ":WGEN<w>:FUNCtion" on page 1197

:WGEN<w>:FREQuency

(see page 1292)

Command Syntax

```
:WGEN<w>:FREQuency <frequency>

<w> ::= 1 to (# WaveGen outputs) in NR1 format

<frequency> ::= frequency in Hz in NR3 format
```

For all waveforms except Noise and DC, the :WGEN<w>:FREQuency command specifies the frequency of the waveform.

You can also specify the frequency indirectly using the :WGEN<w>:PERiod command.

Query Syntax

```
:WGEN<w>:FREQuency?
```

The :WGEN<w>:FREQuency? query returns the currently set waveform generator frequency.

Return Format

```
<frequency><NL>

<frequency> ::= frequency in Hz in NR3 format
```

See Also

- "Introduction to :WGEN<w> Commands" on page 1184
- ":WGEN<w>:FUNCtion" on page 1197
- ":WGEN<w>:PERiod" on page 1216

:WGEN<w>:FUNCtion

N (see page 1292)

Command Syntax

```
:WGEN<w>:FUNCtion <signal>

<w> ::= 1 to (# WaveGen outputs) in NR1 format

<signal> ::= {SINusoid | SQUare | RAMP | PULSe | DC | NOISe | SINC
             | EXPRise | EXPFall | CARDiac | GAUSsian | ARBitrary}
```

The :WGEN<w>:FUNCtion command selects the type of waveform:

Waveform Type	Characteristics	Frequency Range	Max. Amplitude (High-Z)[1]	Offset Range (High-Z)[1]
SINusoid	Use these commands to set the sine signal parameters: ▪ ":WGEN<w>:FREQuency" on page 1196 ▪ ":WGEN<w>:PERiod" on page 1216 ▪ ":WGEN<w>:VOLTage" on page 1218 ▪ ":WGEN<w>:VOLTage:OFFSet" on page 1221 ▪ ":WGEN<w>:VOLTage:HIGH" on page 1219 ▪ ":WGEN<w>:VOLTage:LOW" on page 1220	10 mHz to 100 MHz	2 mVpp to 10 Vpp	When amplitude <50 mV: ±(800 mV - 0.5 * amplitude) When amplitude ≥50 mV: ±(8 V - 0.5 * amplitude)
SQUare	Use these commands to set the square wave signal parameters: ▪ ":WGEN<w>:FREQuency" on page 1196 ▪ ":WGEN<w>:PERiod" on page 1216 ▪ ":WGEN<w>:VOLTage" on page 1218 ▪ ":WGEN<w>:VOLTage:OFFSet" on page 1221 ▪ ":WGEN<w>:VOLTage:HIGH" on page 1219 ▪ ":WGEN<w>:VOLTage:LOW" on page 1220 ▪ ":WGEN<w>:FUNCtion:SQUare:DCYCle" on page 1204 The duty cycle can be adjusted from 20% to 80%.	10 mHz to 50 MHz	2 mVpp to 10 Vpp	When amplitude <50 mV: ±(800 mV - 0.5 * amplitude) When amplitude ≥50 mV: ±(8 V - 0.5 * amplitude)

Waveform Type	Characteristics	Frequency Range	Max. Amplitude (High-Z)[1]	Offset Range (High-Z)[1]
RAMP	Use these commands to set the ramp signal parameters: ▪ ":WGEN<w>:FREQuency" on page 1196 ▪ ":WGEN<w>:PERiod" on page 1216 ▪ ":WGEN<w>:VOLTage" on page 1218 ▪ ":WGEN<w>:VOLTage:OFFSet" on page 1221 ▪ ":WGEN<w>:VOLTage:HIGH" on page 1219 ▪ ":WGEN<w>:VOLTage:LOW" on page 1220 ▪ ":WGEN<w>:FUNCtion:RAMP:SYMMetry" on page 1203 Symmetry represents the amount of time per cycle that the ramp waveform is rising and can be adjusted from 0% to 100%.	300 mHz to 5 MHz	2 mVpp to 10 Vpp	When amplitude <50 mV: ±(800 mV - 0.5 * amplitude) When amplitude ≥50 mV: ±(8 V - 0.5 * amplitude)
PULSe	Use these commands to set the pulse signal parameters: ▪ ":WGEN<w>:FREQuency" on page 1196 ▪ ":WGEN<w>:PERiod" on page 1216 ▪ ":WGEN<w>:VOLTage" on page 1218 ▪ ":WGEN<w>:VOLTage:OFFSet" on page 1221 ▪ ":WGEN<w>:VOLTage:HIGH" on page 1219 ▪ ":WGEN<w>:VOLTage:LOW" on page 1220 ▪ ":WGEN<w>:FUNCtion:PULSe:WIDTh" on page 1202 For frequencies of 101.6 Hz and lower, the minimum pulse width is 300 ns. For frequencies of 6 kHz and higher, the minimum pulse width is 5 ns. For frequencies between 101.6 Hz and 6 kHz, the minimum pulse width goes linearly from 300 ns to 5 ns. The pulse width can be adjusted from the minimum to the period minus the minimum.	10 mHz to 50 MHz.	2 mVpp to 10 Vpp	When amplitude <50 mV: ±(800 mV - 0.5 * amplitude) When amplitude ≥50 mV: ±(8 V - 0.5 * amplitude)

Waveform Type	Characteristics	Frequency Range	Max. Amplitude (High-Z)[1]	Offset Range (High-Z)[1]
DC	Use this command to set the DC level: ▪ ":WGEN<w>:VOLTage:OFFSet" on page 1221 Use this command to set the DC output mode: ▪ ":WGEN<w>:DCMode" on page 1195	n/a	n/a	±8.00 V or ±1.00 V, depending on the DC output mode selection
NOISe	Use these commands to set the noise signal parameters: ▪ ":WGEN<w>:VOLTage" on page 1218 ▪ ":WGEN<w>:VOLTage:OFFSet" on page 1221 ▪ ":WGEN<w>:VOLTage:HIGH" on page 1219 ▪ ":WGEN<w>:VOLTage:LOW" on page 1220	n/a	2 mVpp to 10 Vpp	When amplitude <50 mV: ±(800 mV - 0.5 * amplitude) When amplitude ≥50 mV: ±(8 V - 0.5 * amplitude)
SINC	Use these commands to set the sine cardinal signal parameters: ▪ ":WGEN<w>:FREQuency" on page 1196 ▪ ":WGEN<w>:PERiod" on page 1216 ▪ ":WGEN<w>:VOLTage" on page 1218 ▪ ":WGEN<w>:VOLTage:OFFSet" on page 1221	300 mHz to 5 MHz	2 mVpp to 10 Vpp	When amplitude <50 mV: ±(400 mV - 0.25 * amplitude) When amplitude ≥50 mV: ±(4 V - 0.25 * amplitude)
EXPRise	Use these commands to set the exponential rise signal parameters: ▪ ":WGEN<w>:FREQuency" on page 1196 ▪ ":WGEN<w>:PERiod" on page 1216 ▪ ":WGEN<w>:VOLTage" on page 1218 ▪ ":WGEN<w>:VOLTage:OFFSet" on page 1221 ▪ ":WGEN<w>:VOLTage:HIGH" on page 1219 ▪ ":WGEN<w>:VOLTage:LOW" on page 1220	300 mHz to 5 MHz	2 mVpp to 10 Vpp	When amplitude <50 mV: ±(800 mV - 0.5 * amplitude) When amplitude ≥50 mV: ±(8 V - 0.5 * amplitude)

Waveform Type	Characteristics	Frequency Range	Max. Amplitude (High-Z)[1]	Offset Range (High-Z)[1]
EXPFall	Use these commands to set the exponential fall signal parameters: ▪ ":WGEN<w>:FREQuency" on page 1196 ▪ ":WGEN<w>:PERiod" on page 1216 ▪ ":WGEN<w>:VOLTage" on page 1218 ▪ ":WGEN<w>:VOLTage:OFFSet" on page 1221 ▪ ":WGEN<w>:VOLTage:HIGH" on page 1219 ▪ ":WGEN<w>:VOLTage:LOW" on page 1220	300 mHz to 5 MHz	2 mVpp to 10 Vpp	When amplitude <50 mV: ±(800 mV - 0.5 * amplitude) When amplitude ≥50 mV: ±(8 V - 0.5 * amplitude)
CARDiac	Use these commands to set the cardiac signal parameters: ▪ ":WGEN<w>:FREQuency" on page 1196 ▪ ":WGEN<w>:PERiod" on page 1216 ▪ ":WGEN<w>:VOLTage" on page 1218 ▪ ":WGEN<w>:VOLTage:OFFSet" on page 1221	300 mHz to 5 MHz	2 mVpp to 10 Vpp	When amplitude <50 mV: ±(400 mV - 0.25 * amplitude) When amplitude ≥50 mV: ±(4 V - 0.25 * amplitude)
GAUSsian	Use these commands to set the gaussian pulse signal parameters: ▪ ":WGEN<w>:FREQuency" on page 1196 ▪ ":WGEN<w>:PERiod" on page 1216 ▪ ":WGEN<w>:VOLTage" on page 1218 ▪ ":WGEN<w>:VOLTage:OFFSet" on page 1221	300 mHz to 5 MHz	2 mVpp to 8 Vpp	When amplitude <50 mV: ±(400 mV - 0.25 * amplitude) When amplitude ≥50 mV: ±(4 V - 0.25 * amplitude)

Waveform Type	Characteristics	Frequency Range	Max. Amplitude (High-Z)[1]	Offset Range (High-Z)[1]
ARBitrary	Use these commands to set the arbitrary signal parameters: ▪ ":WGEN<w>:FREQuency" on page 1196 ▪ ":WGEN<w>:PERiod" on page 1216 ▪ ":WGEN<w>:VOLTage" on page 1218 ▪ ":WGEN<w>:VOLTage:OFFSet" on page 1221 ▪ ":WGEN<w>:VOLTage:HIGH" on page 1219 ▪ ":WGEN<w>:VOLTage:LOW" on page 1220	300 mHz to 5 MHz	2 mVpp to 10 Vpp	When amplitude <50 mV: ±(800 mV - 0.5 * amplitude) When amplitude ≥50 mV: ±(8 V - 0.5 * amplitude)
[1]When the output load is 50 Ω, these values are halved.				

Query Syntax

```
:WGEN<w>:FUNCtion?
```

The :WGEN<w>:FUNCtion? query returns the currently selected signal type.

Return Format

```
<signal><NL>

<signal> ::= {SIN | SQU | RAMP | PULS | DC | NOIS | SINC | EXPR | EXPF
              | CARD | GAUS | ARB}
```

See Also

- "Introduction to :WGEN<w> Commands" on page 1184
- ":WGEN<w>:MODulation:NOISe" on page 1210

:WGEN<w>:FUNCtion:PULSe:WIDTh

 (see page 1292)

Command Syntax

```
:WGEN<w>:FUNCtion:PULSe:WIDTh <width>

<w> ::= 1 to (# WaveGen outputs) in NR1 format

<width> ::= pulse width in seconds in NR3 format
```

For Pulse waveforms, the :WGEN<w>:FUNCtion:PULSe:WIDTh command specifies the width of the pulse.

The pulse width can be adjusted from 20 ns to the period minus 20 ns.

Query Syntax

```
:WGEN<w>:FUNCtion:PULSe:WIDTh?
```

The :WGEN<w>:FUNCtion:PULSe:WIDTh? query returns the currently set pulse width.

Return Format

```
<width><NL>

<width> ::= pulse width in seconds in NR3 format
```

See Also

- "Introduction to :WGEN<w> Commands" on page 1184
- ":WGEN<w>:FUNCtion" on page 1197

:WGEN<w>:FUNCtion:RAMP:SYMMetry

N (see page 1292)

Command Syntax

```
:WGEN<w>:FUNCtion:RAMP:SYMMetry <percent>

<w> ::= 1 to (# WaveGen outputs) in NR1 format

<percent> ::= symmetry percentage from 0% to 100% in NR1 format
```

For Ramp waveforms, the :WGEN<w>:FUNCtion:RAMP:SYMMetry command specifies the symmetry of the waveform.

Symmetry represents the amount of time per cycle that the ramp waveform is rising.

Query Syntax

```
:WGEN<w>:FUNCtion:RAMP:SYMMetry?
```

The :WGEN<w>:FUNCtion:RAMP:SYMMetry? query returns the currently set ramp symmetry.

Return Format

```
<percent><NL>

<percent> ::= symmetry percentage from 0% to 100% in NR1 format
```

See Also

- "Introduction to :WGEN<w> Commands" on page 1184
- ":WGEN<w>:FUNCtion" on page 1197

:WGEN<w>:FUNCtion:SQUare:DCYCle

(see page 1292)

Command Syntax

```
:WGEN<w>:FUNCtion:SQUare:DCYCle <percent>

<w> ::= 1 to (# WaveGen outputs) in NR1 format

<percent> ::= duty cycle percentage from 20% to 80% in NR1 format
```

For Square waveforms, the :WGEN<w>:FUNCtion:SQUare:DCYCle command specifies the square wave duty cycle.

Duty cycle is the percentage of the period that the waveform is high.

Query Syntax

```
:WGEN<w>:FUNCtion:SQUare:DCYCle?
```

The :WGEN<w>:FUNCtion:SQUare:DCYCle? query returns the currently set square wave duty cycle.

Return Format

```
<percent><NL>

<percent> ::= duty cycle percentage from 20% to 80% in NR1 format
```

See Also

- "Introduction to :WGEN<w> Commands" on page 1184
- ":WGEN<w>:FUNCtion" on page 1197

:WGEN<w>:MODulation:AM:DEPTh

N (see page 1292)

Command Syntax

```
:WGEN<w>:MODulation:AM:DEPTh <percent>

<w> ::= 1 in NR1 format

<percent> ::= AM depth percentage from 0% to 100% in NR1 format
```

The :WGEN<w>:MODulation:AM:DEPTh command specifies the amount of amplitude modulation.

AM Depth refers to the portion of the amplitude range that will be used by the modulation. For example, a depth setting of 80% causes the output amplitude to vary from 10% to 90% (90% – 10% = 80%) of the original amplitude as the modulating signal goes from its minimum to maximum amplitude.

Query Syntax

```
:WGEN<w>:MODulation:AM:DEPTh?
```

The :WGEN<w>:MODulation:AM:DEPTh? query returns the AM depth percentage setting.

Return Format

```
<percent><NL>

<percent> ::= AM depth percentage from 0% to 100% in NR1 format
```

See Also

- ":WGEN<w>:MODulation:AM:FREQuency" on page 1206
- ":WGEN<w>:MODulation:FM:DEViation" on page 1207
- ":WGEN<w>:MODulation:FM:FREQuency" on page 1208
- ":WGEN<w>:MODulation:FUNCtion" on page 1209
- ":WGEN<w>:MODulation:STATe" on page 1211
- ":WGEN<w>:MODulation:TYPE" on page 1212

:WGEN<w>:MODulation:AM:FREQuency

 (see page 1292)

Command Syntax

```
:WGEN<w>:MODulation:AM:FREQuency <frequency>

<w> ::= 1 in NR1 format

<frequency> ::= modulating waveform frequency in Hz in NR3 format
```

The :WGEN<w>:MODulation:AM:FREQuency command specifies the frequency of the modulating signal.

Query Syntax

```
:WGEN<w>:MODulation:AM:FREQuency?
```

The :WGEN<w>:MODulation:AM:FREQuency? query returns the frequency of the modulating signal.

Return Format

```
<frequency><NL>

<frequency> ::= modulating waveform frequency in Hz in NR3 format
```

See Also

- ":WGEN<w>:MODulation:AM:DEPTh" on page 1205
- ":WGEN<w>:MODulation:FM:DEViation" on page 1207
- ":WGEN<w>:MODulation:FM:FREQuency" on page 1208
- ":WGEN<w>:MODulation:FUNCtion" on page 1209
- ":WGEN<w>:MODulation:STATe" on page 1211
- ":WGEN<w>:MODulation:TYPE" on page 1212

:WGEN<w>:MODulation:FM:DEViation

N (see page 1292)

Command Syntax

```
:WGEN<w>:MODulation:FM:DEViation <frequency>

<w> ::= 1 in NR1 format

<frequency> ::= frequency deviation in Hz in NR3 format
```

The :WGEN<w>:MODulation:FM:DEViation command specifies the frequency deviation from the original carrier signal frequency.

When the modulating signal is at its maximum amplitude, the output frequency is the carrier signal frequency plus the deviation amount, and when the modulating signal is at its minimum amplitude, the output frequency is the carrier signal frequency minus the deviation amount.

The frequency deviation cannot be greater than the original carrier signal frequency.

Also, the sum of the original carrier signal frequency and the frequency deviation must be less than or equal to the maximum frequency for the selected waveform generator function plus 100 kHz.

Query Syntax

```
:WGEN<w>:MODulation:FM:DEViation?
```

The :WGEN<w>:MODulation:FM:DEViation? query returns the frequency deviation setting.

Return Format

```
<frequency><NL>

<frequency> ::= frequency deviation in Hz in NR3 format
```

See Also

- ":WGEN<w>:MODulation:AM:DEPTh" on page 1205
- ":WGEN<w>:MODulation:AM:FREQuency" on page 1206
- ":WGEN<w>:MODulation:FM:FREQuency" on page 1208
- ":WGEN<w>:MODulation:FUNCtion" on page 1209
- ":WGEN<w>:MODulation:STATe" on page 1211
- ":WGEN<w>:MODulation:TYPE" on page 1212

:WGEN<w>:MODulation:FM:FREQuency

(see page 1292)

Command Syntax

```
:WGEN<w>:MODulation:FM:FREQuency <frequency>

<w> ::= 1 in NR1 format

<frequency> ::= modulating waveform frequency in Hz in NR3 format
```

The :WGEN<w>:MODulation:FM:FREQuency command specifies the frequency of the modulating signal.

Query Syntax

```
:WGEN<w>:MODulation:FM:FREQuency?
```

The :WGEN<w>:MODulation:FM:FREQuency? query returns the frequency of the modulating signal.

Return Format

```
<frequency><NL>

<frequency> ::= modulating waveform frequency in Hz in NR3 format
```

See Also

- ":WGEN<w>:MODulation:AM:DEPTh" on page 1205
- ":WGEN<w>:MODulation:AM:FREQuency" on page 1206
- ":WGEN<w>:MODulation:FM:DEViation" on page 1207
- ":WGEN<w>:MODulation:FUNCtion" on page 1209
- ":WGEN<w>:MODulation:STATe" on page 1211
- ":WGEN<w>:MODulation:TYPE" on page 1212

:WGEN<w>:MODulation:FUNCtion

N (see page 1292)

Command Syntax

```
:WGEN<w>:MODulation:FUNCtion <shape>

<w> ::= 1 in NR1 format

<shape> ::= {SINusoid}
```

The :WGEN<w>:MODulation:FUNCtion command specifies the shape of the modulating signal.

This command applies to AM and FM modulation.

Query Syntax

```
:WGEN<w>:MODulation:FUNCtion?
```

The :WGEN<w>:MODulation:FUNCtion? query returns the specified modulating signal shape.

Return Format

```
<shape><NL>

<shape> ::= {SIN}
```

See Also

- ":WGEN<w>:MODulation:AM:DEPTh" on page 1205
- ":WGEN<w>:MODulation:AM:FREQuency" on page 1206
- ":WGEN<w>:MODulation:FM:DEViation" on page 1207
- ":WGEN<w>:MODulation:FM:FREQuency" on page 1208
- ":WGEN<w>:MODulation:STATe" on page 1211
- ":WGEN<w>:MODulation:TYPE" on page 1212

:WGEN<w>:MODulation:NOISe

(see page 1292)

Command Syntax

```
:WGEN<w>:MODulation:NOISe <percent>

<w> ::= 1 in NR1 format

<percent> ::= 0 to 100
```

The :WGEN<w>:MODulation:NOISe command adds noise to the currently selected signal. The sum of the amplitude between the original signal and injected noise is limited to the regular amplitude limit (for example, 5 Vpp in 1 MOhm), so the range for <percent> varies according to current amplitude.

Note that adding noise affects edge triggering on the waveform generator source as well as the waveform generator sync pulse output signal (which can be sent to AUX OUT). This is because the trigger comparator is located after the noise source.

Query Syntax

```
:WGEN<w>:MODulation:NOISe?
```

The :WGEN<w>:MODulation:NOISe query returns the percent of added noise.

Return Format

```
<percent><NL>

<percent> ::= 0 to 100
```

See Also

- ":WGEN<w>:FUNCtion" on page 1197

:WGEN<w>:MODulation:STATe

(see page 1292)

Command Syntax

```
:WGEN<w>:MODulation:STATe <setting>

<w> ::= 1 in NR1 format

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

The :WGEN<w>:MODulation:STATe command enables or disables modulated waveform generator output.

You can enable modulation for all waveform generator function types except pulse, DC, and noise.

Query Syntax

```
:WGEN<w>:MODulation:STATe?
```

The :WGEN<w>:MODulation:STATe? query returns whether the modulated waveform generator output is enabled of disabled.

Return Format

```
<setting><NL>

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

See Also

- ":WGEN<w>:MODulation:AM:DEPTh" on page 1205
- ":WGEN<w>:MODulation:AM:FREQuency" on page 1206
- ":WGEN<w>:MODulation:FM:DEViation" on page 1207
- ":WGEN<w>:MODulation:FM:FREQuency" on page 1208
- ":WGEN<w>:MODulation:FUNCtion" on page 1209
- ":WGEN<w>:MODulation:TYPE" on page 1212

:WGEN<w>:MODulation:TYPE

(see page 1292)

Command Syntax

```
:WGEN<w>:MODulation:TYPE <type>

<w> ::= 1 in NR1 format

<type> ::= {AM | FM}
```

The :WGEN<w>:MODulation:TYPE command selects the modulation type:

- AM (amplitude modulation) — the amplitude of the original carrier signal is modified according to the amplitude of the modulating signal.

  Use the :WGEN<w>:MODulation:AM:FREQuency command to set the modulating signal frequency.

  Use the :WGEN<w>:MODulation:AM:DEPTh command to specify the amount of amplitude modulation.
- FM (frequency modulation) — the frequency of the original carrier signal is modified according to the amplitude of the modulating signal.

  Use the :WGEN<w>:MODulation:FM:FREQuency command to set the modulating signal frequency.

  Use the :WGEN<w>:MODulation:FM:DEViation command to specify the frequency deviation from the original carrier signal frequency.

Query Syntax

```
:WGEN<w>:MODulation:TYPE?
```

The :WGEN<w>:MODulation:TYPE? query returns the selected modulation type.

Return Format

```
<type><NL>

<type> ::= {AM | FM}
```

See Also

- ":WGEN<w>:MODulation:AM:DEPTh" on page 1205
- ":WGEN<w>:MODulation:AM:FREQuency" on page 1206
- ":WGEN<w>:MODulation:FM:DEViation" on page 1207
- ":WGEN<w>:MODulation:FM:FREQuency" on page 1208
- ":WGEN<w>:MODulation:FUNCtion" on page 1209
- ":WGEN<w>:MODulation:STATe" on page 1211

:WGEN<w>:OUTPut

 (see page 1292)

Command Syntax

```
:WGEN<w>:OUTPut <on_off>

<w> ::= 1 to (# WaveGen outputs) in NR1 format

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

The :WGEN<w>:OUTPut command specifies whether the waveform generator signal output is ON (1) or OFF (0).

Query Syntax

```
:WGEN<w>:OUTPut?
```

The :WGEN<w>:OUTPut? query returns the current state of the waveform generator output setting.

Return Format

```
<on_off><NL>

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

See Also

- "Introduction to :WGEN<w> Commands" on page 1184

:WGEN<w>:OUTPut:LOAD

 (see page 1292)

Command Syntax

```
:WGEN<w>:OUTPut:LOAD <impedance>

<w> ::= 1 to (# WaveGen outputs) in NR1 format

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

The :WGEN<w>:OUTPut:LOAD command selects the expected output load impedance.

The output impedance of the Gen Out BNC is fixed at 50 ohms. However, the output load selection lets the waveform generator display the correct amplitude and offset levels for the expected output load.

If the actual load impedance is different than the selected value, the displayed amplitude and offset levels will be incorrect.

Query Syntax

```
:WGEN<w>:OUTPut:LOAD?
```

The :WGEN<w>:OUTPut:LOAD? query returns the current expected output load impedance.

Return Format

```
<impedance><NL>

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

See Also

- "Introduction to :WGEN<w> Commands" on page 1184

:WGEN<w>:OUTPut:POLarity

 (see page 1292)

Command Syntax

```
:WGEN<w>:OUTPut:POLarity <polarity>

<w> ::= 1 to (# WaveGen outputs) in NR1 format

<polarity> ::= {NORMal | INVerted}
```

The :WGEN<w>:OUTPut:POLarity command specifies whether the waveform generator output is inverted..

Query Syntax

```
:WGEN<w>:OUTPut:POLarity?
```

The :WGEN<w>:OUTPut:POLarity? query returns the specified output polarity.

Return Format

```
<polarity><NL>

<polarity> ::= {NORM | INV}
```

See Also

- "Introduction to :WGEN<w> Commands" on page 1184

:WGEN<w>:PERiod

(see page 1292)

Command Syntax

```
:WGEN<w>:PERiod <period>

<w> ::= 1 to (# WaveGen outputs) in NR1 format

<period> ::= period in seconds in NR3 format
```

For all waveforms except Noise and DC, the :WGEN<w>:PERiod command specifies the period of the waveform.

You can also specify the period indirectly using the :WGEN<w>:FREQuency command.

Query Syntax

```
:WGEN<w>:PERiod?
```

The :WGEN<w>:PERiod? query returns the currently set waveform generator period.

Return Format

```
<period><NL>

<period> ::= period in seconds in NR3 format
```

See Also

- "Introduction to :WGEN<w> Commands" on page 1184
- ":WGEN<w>:FUNCtion" on page 1197
- ":WGEN<w>:FREQuency" on page 1196

:WGEN<w>:RST

 (see page 1292)

Command Syntax

```
:WGEN<w>:RST

<w> ::= 1 to (# WaveGen outputs) in NR1 format
```

The :WGEN<w>:RST command restores the waveform generator factory default settings (1 kHz sine wave, 500 mVpp, 0 V offset).

See Also

- "Introduction to :WGEN<w> Commands" on page 1184
- ":WGEN<w>:FUNCtion" on page 1197
- ":WGEN<w>:FREQuency" on page 1196

:WGEN<w>:VOLTage

N (see page 1292)

Command Syntax

```
:WGEN<w>:VOLTage <amplitude>

<w> ::= 1 to (# WaveGen outputs) in NR1 format

<amplitude> ::= amplitude in volts in NR3 format
```

For all waveforms except DC, the :WGEN<w>:VOLTage command specifies the waveform's amplitude. Use the :WGEN<w>:VOLTage:OFFSet command to specify the offset voltage or DC level.

You can also specify the amplitude and offset indirectly using the :WGEN<w>:VOLTage:HIGH and :WGEN<w>:VOLTage:LOW commands. For example, an amplitude of 5 V and an offset of 1 V is the same as a high-level voltage of 4 V and a low-level voltage of -1 V.

NOTE When the amplitude is below 40 mV, the offset is limited to ±500 mV.

Query Syntax

```
:WGEN<w>:VOLTage?
```

The :WGEN<w>:VOLTage? query returns the currently specified waveform amplitude.

Return Format

```
<amplitude><NL>

<amplitude> ::= amplitude in volts in NR3 format
```

See Also

- "Introduction to :WGEN<w> Commands" on page 1184
- ":WGEN<w>:FUNCtion" on page 1197
- ":WGEN<w>:VOLTage:OFFSet" on page 1221
- ":WGEN<w>:VOLTage:HIGH" on page 1219
- ":WGEN<w>:VOLTage:LOW" on page 1220

:WGEN<w>:VOLTage:HIGH

N (see page 1292)

Command Syntax

```
:WGEN<w>:VOLTage:HIGH <high>

<w> ::= 1 to (# WaveGen outputs) in NR1 format

<high> ::= high-level voltage in volts, in NR3 format
```

For all waveforms except DC, the :WGEN<w>:VOLTage:HIGH command specifies the waveform's high-level voltage. Use the :WGEN<w>:VOLTage:LOW command to specify the low-level voltage.

You can also specify the high-level and low-level voltages indirectly using the :WGEN<w>:VOLTage and :WGEN<w>:VOLTage:OFFSet commands. For example, a high-level voltage of 4 V and a low-level voltage of -1 V is the same as an amplitude of 5 V and an offset of 1 V.

NOTE When the amplitude is below 40 mV, the offset is limited to ±500 mV.

Query Syntax

```
:WGEN<w>:VOLTage:HIGH?
```

The :WGEN<w>:VOLTage:HIGH? query returns the currently specified waveform high-level voltage.

Return Format

```
<high><NL>

<high> ::= high-level voltage in volts, in NR3 format
```

See Also

- "Introduction to :WGEN<w> Commands" on page 1184
- ":WGEN<w>:FUNCtion" on page 1197
- ":WGEN<w>:VOLTage:LOW" on page 1220
- ":WGEN<w>:VOLTage" on page 1218
- ":WGEN<w>:VOLTage:OFFSet" on page 1221

:WGEN<w>:VOLTage:LOW

N (see page 1292)

Command Syntax

```
:WGEN<w>:VOLTage:LOW <low>

<w> ::= 1 to (# WaveGen outputs) in NR1 format

<low> ::= low-level voltage in volts, in NR3 format
```

For all waveforms except DC, the :WGEN<w>:VOLTage:LOW command specifies the waveform's low-level voltage. Use the :WGEN<w>:VOLTage:HIGH command to specify the high-level voltage.

You can also specify the high-level and low-level voltages indirectly using the :WGEN<w>:VOLTage and :WGEN<w>:VOLTage:OFFSet commands. For example, a high-level voltage of 4 V and a low-level voltage of -1 V is the same as an amplitude of 5 V and an offset of 1 V.

NOTE When the amplitude is below 40 mV, the offset is limited to ±500 mV.

Query Syntax

```
:WGEN<w>:VOLTage:LOW?
```

The :WGEN<w>:VOLTage:LOW? query returns the currently specified waveform low-level voltage.

Return Format

```
<low><NL>

<low> ::= low-level voltage in volts, in NR3 format
```

See Also

- "Introduction to :WGEN<w> Commands" on page 1184
- ":WGEN<w>:FUNCtion" on page 1197
- ":WGEN<w>:VOLTage:LOW" on page 1220
- ":WGEN<w>:VOLTage" on page 1218
- ":WGEN<w>:VOLTage:OFFSet" on page 1221

:WGEN<w>:VOLTage:OFFSet

(see page 1292)

Command Syntax

```
:WGEN<w>:VOLTage:OFFSet <offset>

<w> ::= 1 to (# WaveGen outputs) in NR1 format

<offset> ::= offset in volts in NR3 format
```

The :WGEN<w>:VOLTage:OFFSet command specifies the waveform's offset voltage or the DC level. Use the :WGEN<w>:VOLTage command to specify the amplitude.

You can also specify the amplitude and offset indirectly using the :WGEN<w>:VOLTage:HIGH and :WGEN<w>:VOLTage:LOW commands. For example, an amplitude of 5 V and an offset of 1 V is the same as a high-level voltage of 4 V and a low-level voltage of -1 V.

NOTE When the amplitude is below 40 mV, the offset is limited to ±500 mV.

Query Syntax

```
:WGEN<w>:VOLTage:OFFSet?
```

The :WGEN<w>:VOLTage:OFFSet? query returns the currently specified waveform offset voltage.

Return Format

```
<offset><NL>

<offset> ::= offset in volts in NR3 format
```

See Also

- "Introduction to :WGEN<w> Commands" on page 1184
- ":WGEN<w>:FUNCtion" on page 1197
- ":WGEN<w>:VOLTage" on page 1218
- ":WGEN<w>:VOLTage:HIGH" on page 1219
- ":WGEN<w>:VOLTage:LOW" on page 1220

38 :WMEMory<r> Commands

Control reference waveforms.

Table 145 :WMEMory<r> Commands Summary

Command	Query	Options and Query Returns
:WMEMory<r>:CLEar (see page 1225)	n/a	<r> ::= 1 to (# ref waveforms) in NR1 format
:WMEMory<r>:DISPlay {{0 | OFF} | {1 | ON}} (see page 1226)	:WMEMory<r>:DISPlay? (see page 1226)	<r> ::= 1 to (# ref waveforms) in NR1 format {0 | 1}
:WMEMory<r>:LABel <string> (see page 1227)	:WMEMory<r>:LABel? (see page 1227)	<r> ::= 1 to (# ref waveforms) in NR1 format <string> ::= any series of 32 or less ASCII characters enclosed in quotation marks
:WMEMory<r>:SAVE <source> (see page 1228)	n/a	<r> ::= 1 to (# ref waveforms) in NR1 format <source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format NOTE: Math functions whose x-axis is not frequency can be saved as reference waveforms.
:WMEMory<r>:SKEW <skew> (see page 1229)	:WMEMory<r>:SKEW? (see page 1229)	<r> ::= 1 to (# ref waveforms) in NR1 format <skew> ::= time in seconds in NR3 format

Table 145 :WMEMory<r> Commands Summary (continued)

Command	Query	Options and Query Returns
:WMEMory<r>:YOFFset <offset>[suffix] (see page 1230)	:WMEMory<r>:YOFFset? (see page 1230)	<r> ::= 1 to (# ref waveforms) in NR1 format <offset> ::= vertical offset value in NR3 format [suffix] ::= {V | mV}
:WMEMory<r>:YRANge <range> (see page 1231)	:WMEMory<r>:YRANge? (see page 1231)	<r> ::= 1 to (# ref waveforms) in NR1 format <range> ::= vertical full-scale range value in NR3 format
:WMEMory<r>:YSCale <scale> (see page 1232)	:WMEMory<r>:YSCale? (see page 1232)	<r> ::= 1 to (# ref waveforms) in NR1 format <scale> ::= vertical units per division value in NR3 format

:WMEMory<r>:CLEar

N (see page 1292)

Command Syntax

```
:WMEMory<r>:CLEar

<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :WMEMory<r>:CLEar command clears the specified reference waveform location.

See Also

- Chapter 38, ":WMEMory<r> Commands," starting on page 1223
- ":WMEMory<r>:SAVE" on page 1228
- ":WMEMory<r>:DISPlay" on page 1226

:WMEMory<r>:DISPlay

 (see page 1292)

Command Syntax

```
:WMEMory<r>:DISPlay <on_off>

<r> ::= 1 to (# ref waveforms) in NR1 format

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

The :WMEMory<r>:DISPlay command turns the display of the specified reference waveform on or off.

There are two reference waveform locations, but only one reference waveform can be displayed at a time. That means, if :WMEMory1:DISPlay is ON, sending the :WMEMory2:DISPlay ON command will automatically set :WMEMory1:DISPlay OFF.

Query Syntax

```
:WMEMory<r>:DISPlay?
```

The :WMEMory<r>:DISPlay? query returns the current display setting for the reference waveform.

Return Format

```
<on_off><NL>

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

See Also

- Chapter 38, ":WMEMory<r> Commands," starting on page 1223
- ":WMEMory<r>:CLEar" on page 1225
- ":WMEMory<r>:LABel" on page 1227

:WMEMory<r>:LABel

N (see page 1292)

Command Syntax

```
:WMEMory<r>:LABel <string>

<r> ::= 1 to (# ref waveforms) in NR1 format

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

NOTE Label strings are 10 characters or less, and may contain any commonly used ASCII characters. Labels with more than 10 characters are truncated to 10 characters. Lower case characters are converted to upper case.

The :WMEMory<r>:LABel command sets the reference waveform label to the string that follows.

Setting a label for a reference waveform also adds the name to the label list in non-volatile memory (replacing the oldest label in the list).

Query Syntax

```
:WMEMory<r>:LABel?
```

The :WMEMory<r>:LABel? query returns the label associated with a particular reference waveform.

Return Format

```
<string><NL>

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

See Also

- Chapter 38, ":WMEMory<r> Commands," starting on page 1223
- ":WMEMory<r>:DISPlay" on page 1226

:WMEMory<r>:SAVE

N (see page 1292)

Command Syntax

```
:WMEMory<r>:SAVE <source>

<r> ::= 1 to (# ref waveforms) in NR1 format

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

<n> ::= 1 to (# analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format
```

The :WMEMory<r>:SAVE command copies the analog channel or math function waveform to the specified reference waveform location.

NOTE Math functions whose x-axis is not frequency can be saved as reference waveforms.

See Also

- Chapter 38, ":WMEMory<r> Commands," starting on page 1223
- ":WMEMory<r>:DISPlay" on page 1226

:WMEMory<r>:SKEW

N (see page 1292)

Command Syntax

```
:WMEMory<r>:SKEW <skew>

<r> ::= 1 to (# ref waveforms) in NR1 format

<skew> ::= time in seconds in NR3 format
```

The :WMEMory<r>:SKEW command sets the skew factor for the specified reference waveform.

Query Syntax

```
:WMEMory<r>:SKEW?
```

The :WMEMory<r>:SKEW? query returns the current skew setting for the selected reference waveform.

Return Format

```
<skew><NL>

<skew> ::= time in seconds in NR3 format
```

See Also

- Chapter 38, ":WMEMory<r> Commands," starting on page 1223
- ":WMEMory<r>:DISPlay" on page 1226
- ":WMEMory<r>:YOFFset" on page 1230
- ":WMEMory<r>:YRANge" on page 1231
- ":WMEMory<r>:YSCale" on page 1232

:WMEMory<r>:YOFFset

(see page 1292)

Command Syntax

```
:WMEMory<r>:YOFFset <offset> [<suffix>]

<r> ::= 1 to (# ref waveforms) in NR1 format

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

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

The :WMEMory<r>:YOFFset command sets the value that is represented at center screen for the selected reference waveform.

The range of legal values varies with the value set by the :WMEMory<r>:YRANge or :WMEMory<r>:YSCale 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 affected by the probe attenuation setting.

Query Syntax

```
:WMEMory<r>:YOFFset?
```

The :WMEMory<r>:YOFFset? query returns the current offset value for the selected reference waveform.

Return Format

```
<offset><NL>

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

See Also

- Chapter 38, ":WMEMory<r> Commands," starting on page 1223
- ":WMEMory<r>:DISPlay" on page 1226
- ":WMEMory<r>:YRANge" on page 1231
- ":WMEMory<r>:YSCale" on page 1232
- ":WMEMory<r>:SKEW" on page 1229

:WMEMory<r>:YRANge

N (see page 1292)

Command Syntax

```
:WMEMory<r>:YRANge <range>

<r> ::= 1 to (# ref waveforms) in NR1 format

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

The :WMEMory<r>:YRANge command defines the full-scale vertical axis of the selected reference waveform.

Legal values for the range are copied from the original source waveform (that is, the analog channel or math function waveform that was originally saved as a reference waveform).

Query Syntax

```
:WMEMory<r>:YRANge?
```

The :WMEMory<r>:YRANge? query returns the current full-scale range setting for the specified reference waveform.

Return Format

```
<range><NL>

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

See Also

- Chapter 38, ":WMEMory<r> Commands," starting on page 1223
- ":WMEMory<r>:DISPlay" on page 1226
- ":WMEMory<r>:YOFFset" on page 1230
- ":WMEMory<r>:SKEW" on page 1229
- ":WMEMory<r>:YSCale" on page 1232

:WMEMory<r>:YSCale

(see page 1292)

Command Syntax

```
:WMEMory<r>:YSCale <scale>

<r> ::= 1 to (# ref waveforms) in NR1 format

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

The :WMEMory<r>:YSCale command sets the vertical scale, or units per division, of the selected reference waveform.

Legal values for the scale are copied from the original source waveform (that is, the analog channel or math function waveform that was originally saved as a reference waveform).

Query Syntax

```
:WMEMory<r>:YSCale?
```

The :WMEMory<r>:YSCale? query returns the current scale setting for the specified reference waveform.

Return Format

```
<scale><NL>

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

See Also

- Chapter 38, ":WMEMory<r> Commands," starting on page 1223
- ":WMEMory<r>:DISPlay" on page 1226
- ":WMEMory<r>:YOFFset" on page 1230
- ":WMEMory<r>:YRANge" on page 1231
- ":WMEMory<r>:SKEW" on page 1229

39 Obsolete and Discontinued Commands

Obsolete Commands

Obsolete commands are older forms of commands that are provided to reduce customer rework for existing systems and programs (see"Obsolete Commands" on page 1292).

Obsolete Command	Current Command Equivalent	Behavior Differences

Discontinued Commands

Discontinued commands are commands that were used by previous oscilloscopes, but are not supported by the InfiniiVision HD3-Series oscilloscopes. Listed below are the Discontinued commands and the nearest equivalent command available (if any).

Discontinued Command	Current Command Equivalent	Comments
:ACQuire:AALias?	none	Anti-alaising is always present in the HD3-Series oscilloscopes.
:ACQuire:DAALias	none	There is no option to turn off anti-alaising in the HD3-Series oscilloscopes.
:CHANnel2:SKEW	:CHANnel<n>:PROBe:SKEW (see page 291)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:CHANnel:ACTivity	:ACTivity (see page 210)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:CHANnel:LABel	:CHANnel<n>:LABel (see page 278) or :DIGital<n>:LABel (see page 315)	This was an obsolete command in the 3000G X-Series oscilloscopes.

Discontinued Command	Current Command Equivalent	Comments
:CHANnel:THReshold	:POD<n>:THReshold (see page 673) or :DIGital<d>:THReshold (see page 318)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:CHANnel<n>:BANDwidth	:ACQuire:BANDwidth (see page 232) or :CHANnel<n>:BWLimit (see page 273)	The the HD3-Series oscilloscopes have a configurable global bandwidth limit (see :ACQuire:BANDwidth) and a per-channel 40 MHz bandwidth limit (see :CHANnel:BWLimit).
:CHANnel<n>:INPut	:CHANnel<n>:IMPedance (see page 276)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:CHANnel<n>:PMODe	none	This was an obsolete command in the 3000G X-Series oscilloscopes.
:DIGital<d>:POSition	:DIGital:ORDer[:SET] (see page 313)	The notion of digital channel position locations is not present in the HD3-Series oscilloscopes. You can, however, order digital channels using the :DIGital:ORDer[:SET] (see page 313) command.
:DISPlay:ANNotation<n>:X1Position	:DISPlay:ANNotation<n>:XPOSition (see page 331)	Discontinued command positioned by pixel position, current command positions by percent of grid width.
:DISPlay:ANNotation<n>:Y1Position	:DISPlay:ANNotation<n>:YPOSition (see page 332)	Discontinued command positioned by pixel position, current command positions by percent of grid height.
:DISPlay:CONNect	none	This was an obsolete command in the 3000G X-Series oscilloscopes.
:DISPlay:GRATicule:TYPE	none	This command was for the TV trigger mode which is not available in the HD3-Series oscilloscopes.

Discontinued Command	Current Command Equivalent	Comments
:DISPlay:MENU	none	On earlier InfiniiVision oscilloscopes, this command would display a particular softkey menu. The HD3-Series oscilloscopes do not have softkey menus.
:DISPlay:ORDer	none	This was an obsolete command in the 3000G X-Series oscilloscopes.
:DISPlay:SIDebar	none	On earlier InfiniiVision oscilloscopes, this command would display a particular sidebar dialog box. The HD3-Series oscilloscopes do not have sidebar dialog boxes.
:DISPlay:VECTors	none	On the 3000G X-Series oscilloscopes, lines connecting acquisition points was always on. Because it is also always on in the HD3-Series oscilloscopes, this command is not necessary.
:DVM:FREQuency?	:COUNter<c>:CURRent? (see page 301)	The :DVM:FREQuency? query has been replaced by the new Chapter 12, ":COUNter<c> Commands," starting on page 299.
:ERASe	:DISplay:CLEar (see page 334)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:EXTernal:PMODe	none	This was an obsolete command in the 3000G X-Series oscilloscopes.
:FFT:DMODe	none	FFT display mode options are not available in the HD3-Series oscilloscopes. However, You can enable an FFT Max Function in the user interface.
:FUNCtion:GOFT:OPERation	:FUNCtion1:OPERation (see page 456)	GOFT maps to FUNCtion1. This was an obsolete command in the 3000G X-Series oscilloscopes.

Discontinued Command	Current Command Equivalent	Comments
:FUNCtion:GOFT:SOURce1	:FUNCtion1:SOURce1 (see page 464)	GOFT maps to FUNCtion1. This was an obsolete command in the 3000G X-Series oscilloscopes.
:FUNCtion:GOFT:SOURce2	:FUNCtion1:SOURce2 (see page 466)	GOFT maps to FUNCtion1. This was an obsolete command in the 3000G X-Series oscilloscopes.
:FUNCtion:SOURce	:FUNCtion:SOURce1 (see page 464)	Obsolete command has ADD, SUBTract, and MULTiply parameters; current command has GOFT parameter. This was an obsolete command in the 3000G X-Series oscilloscopes.
:FUNCtion:VIEW	:FUNCtion:DISPlay (see page 431)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:HARDcopy:DESTination	:RECall:FILename (see page 680) :SAVE:FILename (see page 680)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:HARDcopy:FILename	:RECall:FILename (see page 680) :SAVE:FILename (see page 680)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:HARDcopy:GRAYscale	none	This was an obsolete command in the 3000G X-Series oscilloscopes.
:HARDcopy:IGColors	none	This was an obsolete command in the 3000G X-Series oscilloscopes.
:HARDcopy:PDRiver	none	This was an obsolete command in the 3000G X-Series oscilloscopes.
:HWEenable	:STATus:OPERation:HARDware:ENABle (see page 925)	Status registers now use SCPI standard commands and queries for controlling and reading the registers.
:HWERegister:CONDition?	:STATus:OPERation:HARDware:CONDition? (see page 924)	
:HWERegister[:EVENt]?	:STATus:OPERation:HARDware[:EVENt]? (see page 928)	

Discontinued Command	Current Command Equivalent	Comments
:MEASure:ALL	:MEASure:QUICk (see page 589)	There is no "Snapshot All" feature in the HD3-Series oscilloscopes, but there is a "Quick Measure" that will install 10 common measurements on an analog input channel source.
:MEASure:DEFine	:MEASure:DELay:DEFine (see page 555) :MEASure:THResholds:ABSolute (see page 612) :MEASure:THResholds:METHod (see page 613) :MEASure:THResholds:PERCent (see page 614) :MEASure:THResholds:STANdard (see page 615)	The :MEASure:DEFine functionality is provided by the current command equivalents.
:MEASure:LOWer	:MEASure:THResholds:ABSolute (see page 612) :MEASure:THResholds:METHod (see page 613) :MEASure:THResholds:PERCent (see page 614) :MEASure:THResholds:STANdard (see page 615)	The :MEASure:THResholds commands can define absolute values or percentage. This was an obsolete command in the 3000G X-Series oscilloscopes.
:MEASure:TDELta	:MARKer:XDELta (see page 518)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:MEASure:THResholds	:MEASure:THResholds:ABSolute (see page 612) :MEASure:THResholds:METHod (see page 613) :MEASure:THResholds:PERCent (see page 614) :MEASure:THResholds:STANdard (see page 615)	The :MEASure:THResholds commands can define absolute values or percentage. This was an obsolete command in the 3000G X-Series oscilloscopes.
:MEASure:TMAX	:MEASure:XMAX (see page 628)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:MEASure:TMIN	:MEASure:XMIN (see page 629)	This was an obsolete command in the 3000G X-Series oscilloscopes.

Discontinued Command	Current Command Equivalent	Comments
:MEASure:TSTArt	:MARKer:X1Position (see page 513)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:MEASure:TSTOp	:MARKer:X2Position (see page 516)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:MEASure:TVOLt	:MEASure:TVALue (see page 616)	TVALue measures additional values such as db, Vs, etc. This was an obsolete command in the 3000G X-Series oscilloscopes.
:MEASure:UPPer	:MEASure:THResholds:ABSolute (see page 612) :MEASure:THResholds:METHod (see page 613) :MEASure:THResholds:PERCent (see page 614) :MEASure:THResholds:STANdard (see page 615)	The :MEASure:THResholds commands can define absolute values or percentage. This was an obsolete command in the 3000G X-Series oscilloscopes.
:MEASure:VDELta	:MARKer:YDELta (see page 525)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:MEASure:VSTArt	:MARKer:Y1Position (see page 522)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:MEASure:VSTOp	:MARKer:Y2Position (see page 524)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:MEASure:VTIMe	:MEASure:YATX (see page 630)	With :MEASure:YATX, there is a command for installing the measurement on the oscilloscope's display. :MEASure:VTIMe had no command, only a query.
:MTEenable	:STATus:OPERation:MTESt:ENABle (see page 951)	Status registers now use SCPI standard commands and queries for controlling and reading status registers.
:MTERegister[:EVENt]?	:STATus:OPERation:MTESt[:EVENt]? (see page 954)	
:MTESt:AMASk:{SAVE | STORe}	:SAVE:MASK[:STARt] (see page 697)	This was an obsolete command in the 3000G X-Series oscilloscopes.

Discontinued Command	Current Command Equivalent	Comments
:MTESt:AVERage	:ACQuire:TYPE AVERage (see page 247)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:MTESt:AVERage:COUNt	:ACQuire:COUNt (see page 234)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:MTESt:LOAD	:RECall:MASK[:STARt] (see page 682)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:MTESt:RUMode	:MTESt:RMODe (see page 651)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:MTESt:RUMode:SOFailure	:MTESt:RMODe:FACTion:STOP (see page 655)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:MTESt:{STARt | STOP}	:RUN (see page 221) or :STOP (see page 225)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:MTESt:TRIGger:SOURce	:TRIGger Commands (see page 1077)	There are various commands for setting the source with different types of triggers. This was an obsolete command in the 3000G X-Series oscilloscopes.
:OPEE	:STATus:OPERation:ENABle (see page 899)	Status registers now use SCPI standard commands and queries for controlling and reading status registers.
:OPERegister:CONDition?	:STATus:OPERation:CONDition? (see page 898)	
:OPERegister[:EVENt]?	:STATus:OPERation[:EVENt]? (see page 902)	
:OVLenable	:STATus:OPERation:OVERload:ENABle (see page 964)	Status registers now use SCPI standard commands and queries for controlling and reading status registers.
:OVLRegister?	:STATus:OPERation:OVERload[:EVENt]? (see page 967)	
:PRINt?	:DISPlay:DATA? (see page 339)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:SAVE:IMAGe:AREA	none	This was an obsolete command in the 3000G X-Series oscilloscopes.

Discontinued Command	Current Command Equivalent	Comments
:SAVE:IMAGe:PALette	none	Color images are always saved in the HD3-Series oscilloscopes.
:SBUS<n>:CAN:FDSTandard	none	This is no longer an option. The ISO standard is used when decoding or triggering on FD frames.
:SBUS<n>:LIN:SIGNal:DEFinitio n	none	This was an obsolete command in the 3000G X-Series oscilloscopes.
:SYSTem:MENU	none	This was an obsolete command in the 3000G X-Series oscilloscopes.
:SYSTem:PRECision	:SYSTem:PRECision:LENGth (see page 1043)	Precision analysis is no longer a feature to be enabled. Instead, you simply choose the desired analysis record length.
:TRIGger:THReshold	:POD<n>:THReshold (see page 673) or :DIGital<d>:THReshold (see page 318)	This was an obsolete command in the 3000G X-Series oscilloscopes.
:TRIGger:ZONE:STATe	none	There is no longer an overall zone trigger feature on/off state.
:WAVeform:VIEW	none	In the HD3-Series oscilloscopes, all captured data is used.

Discontinued Parameters

Some previous oscilloscope queries returned control setting values of OFF and ON. The InfiniiVision HD3-Series oscilloscopes return only the enumerated values 0 (for off) and 1 (for on).

:MEASure:SCRatch

 (see page 1292)

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 (see page 550) instead.**

:SBUS<n>:SPI:SOURce:DATA

 (see page 1292)

Command Syntax

```
:SBUS<n>:SPI:SOURce:DATA <source>

<source> ::= {CHANnel<n> | DIGital<d>}

<n> ::= 1 to (# analog channels) in NR1 format

<d> ::= 0 to (# digital channels - 1) in NR1 format
```

The :SBUS<n>:SPI:SOURce:DATA command sets the source for the SPI serial MOSI data.

This command is the same as the :SBUS<n>:SPI:SOURce:MOSI command.

Query Syntax

```
:SBUS<n>:SPI:SOURce:DATA?
```

The :SBUS<n>:SPI:SOURce:DATA? query returns the current source for the SPI serial MOSI data.

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 1077
- ":SBUS<n>:SPI:SOURce:MOSI" on page 791
- ":SBUS<n>:SPI:SOURce:MISO" on page 790
- ":SBUS<n>:SPI:SOURce:CLOCk" on page 788
- ":SBUS<n>:SPI:SOURce:FRAMe" on page 789
- ":SBUS<n>:SPI:TRIGger:PATTern:MISO:DATA" on page 792
- ":SBUS<n>:SPI:TRIGger:PATTern:MOSI:DATA" on page 794
- ":SBUS<n>:SPI:TRIGger:PATTern:MISO:WIDTh" on page 793
- ":SBUS<n>:SPI:TRIGger:PATTern:MOSI:WIDTh" on page 795

:TIMebase[:MAIN]:DELay

(see page 1292)

Command Syntax

```
:TIMebase[:MAIN]: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[:MAIN]: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 (see page 1071).

NOTE The :TIMebase[:MAIN]:DELay command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :TIMebase[:MAIN]:POSition command (see page 1067) instead.

Query Syntax

```
:TIMebase[:MAIN]:DELay?
```

The :TIMebase[:MAIN]: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.

' Set time base delay to 0.0.
myScope.WriteString ":TIMebase:DELay 0.0"
```

See complete example programs at: Chapter 44, “Programming Examples,” starting on page 1301

40 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.

-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

A remote operation wants some information, but there is no information available. For example, you may request a stored TIFF image using the :DISPlay:DATA? query, but there may be no image stored.

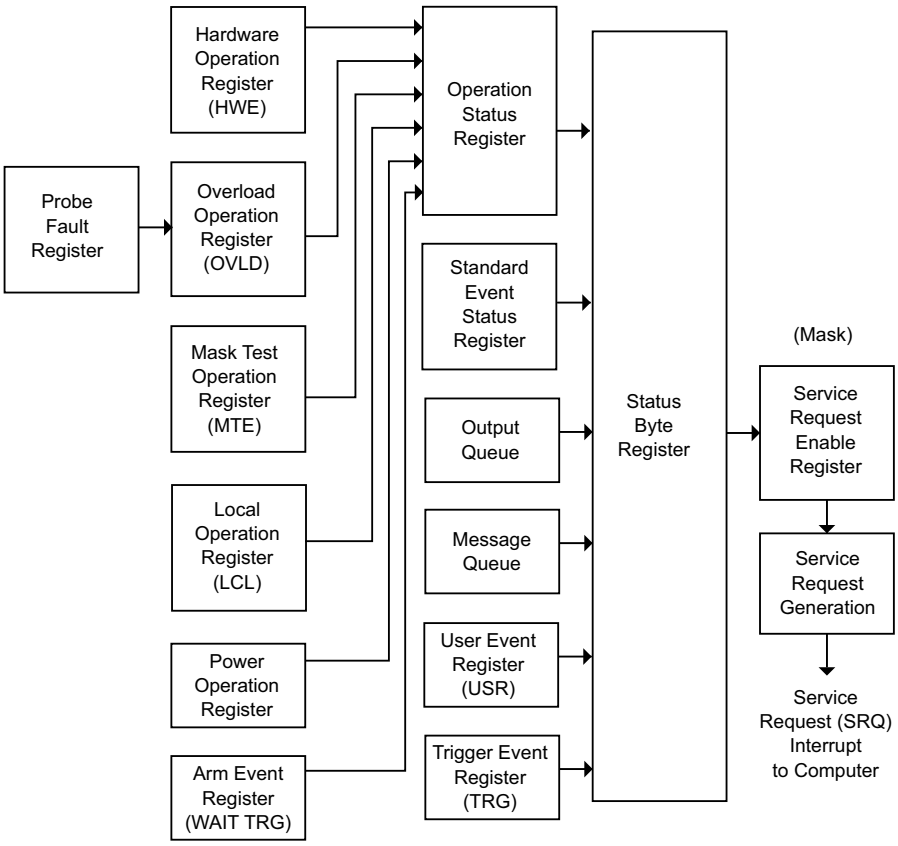
+112, Unknown File Type

+113, Directory Not Supported

41 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 Register and the Overload Operation 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.



- 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 status registers, the summary bit is set if an event in its Condition register, as defined by its Transition Filters, is latched into the Event register and the correcponding bit in its Enable register is set (see "Introduction to :STATus Commands" on page 883). Events latched to an Event register remain set until the register is queried or cleared. Registers are queried 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.

See Also
- "Service Request Enable Register" on page 198
  - "Status Byte Register" on page 200
    - "Trigger Event Register" on page 995
    - "User Event Register" on page 1008
    - "Message Queue" on page 1258
    - "Output Queue" on page 1257
    - "Standard Event Status Register" on page 187
    - "Operation Status Register" on page 889
      - "Arm Event Register" on page 904
      - "Power Operation Register" on page 982
      - "Local Operation Register" on page 930
      - "Mask Test Operation Register" on page 943
      - "Overload Operation Register" on page 956
        - "Probe Fault Register" on page 969
      - "Hardware Operation Register" on page 917

Status Reporting Data Structures

The status register bits are described in the following tables:

- Table 70 – Summary message goes to Status Byte Register bit 6.
  - Table 71
    - Table 129 – Summary message goes to Status Byte Register bit 0 (TRG).
    - Table 131 – Summary message goes to Status Byte Register bit 1 (USR).
    - Table 69 – Summary message goes to Status Byte Register bit 5 (ESB).
    - Table 113 – Summary message goes to Status Byte Register bit 7 (OPER).
      - Table 115 – Summary message goes to Operation Status Register bit 5 (WAIT TRIG).
      - Table 127 – Summary message goes to Operation Status Register bit 7 (Power).
      - Table 119 – Summary message goes to Operation Status Register bit 8 (Lcl).
      - Table 121 – Summary message goes to Operation Status Register bit 9 (MTE).
      - Table 123 – Summary message goes to Operation Status Register bit 11 (OVLD).
        - Table 125 – Summary message goes to Overload Operation Register bit 12 (Channel Probe Fault Summary)
      - Table 117 – Summary message goes to Operation Status Register bit 12 (HWE).

The status registers hirearchy 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.

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 Keysight VISA COM library, the output queue may be read with the FormattedIO488 object's ReadString, ReadNumber, ReadList, or ReadIEEEBlock methods.

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.

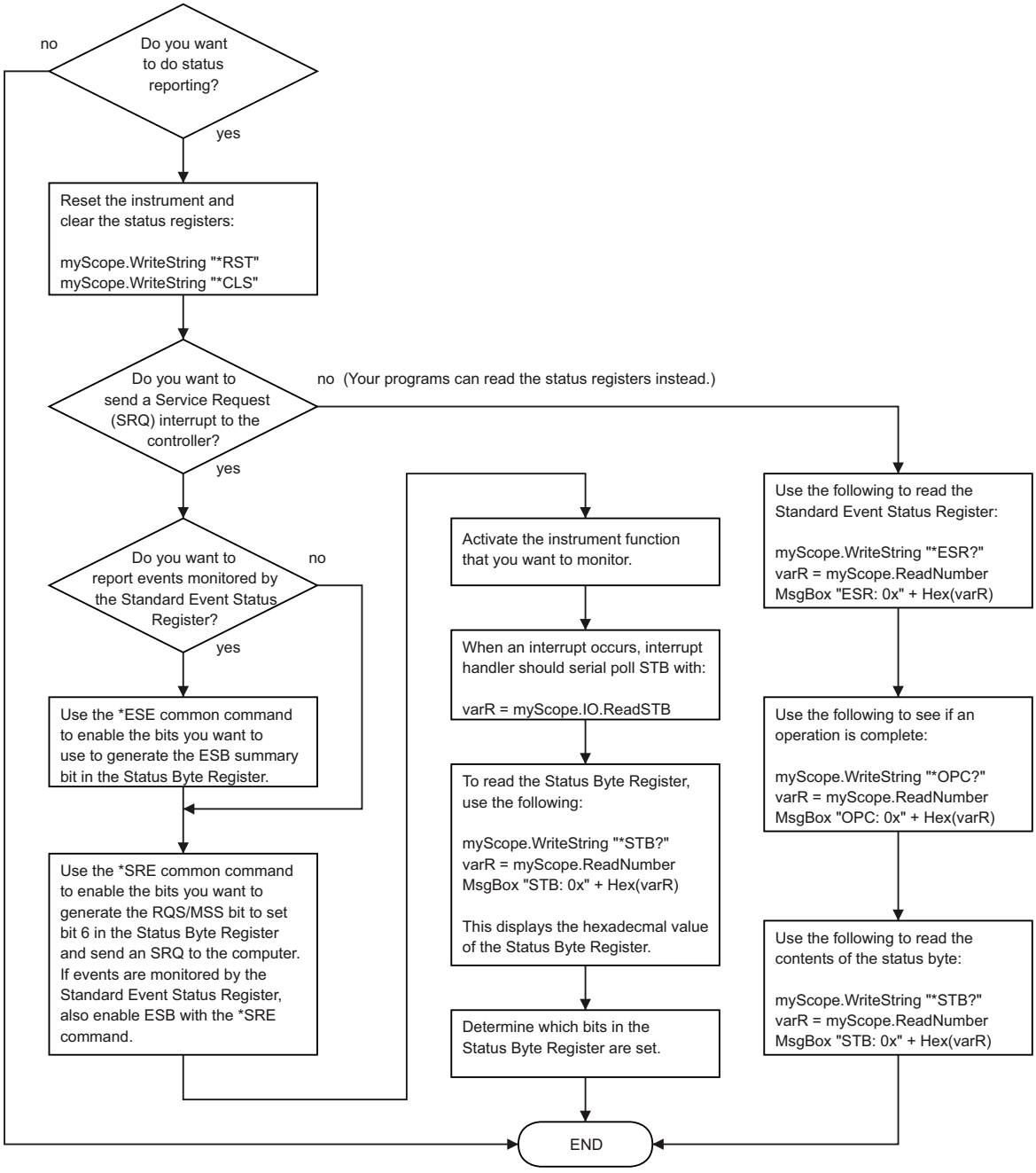
Clearing Registers and Queues

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.

See Also

- "*CLS (Clear Status)" on page 184

Status Reporting Decision Chart



Example: Checking for Armed Status

```
#!python3
# -*- coding: utf-8 -*-

# ********************************************************************
# This script using the Python language (http://www.python.org/) and
# the PyVISA package (http://pyvisa.readthedocs.org/) shows three
# methods to tell whether a Keysight InfiniiVision oscilloscope is
# armed.
# ********************************************************************

# Import modules
# --------------------------------------------------------------------
import sys
import pyvisa
import time

# Initialization constants
# --------------------------------------------------------------------
# Get VISA address from Keysight IO Libraries Connection Expert
VISA_ADDRESS = "USB0::0x2A8D::0x4704::Unset::0::INSTR"
GLOBAL_TOUT =  20000   # IO timeout in milliseconds


# ====================================================================
# Method 1: Query the Armed Event Register with :AER?
# --------------------------------------------------------------------
# This method reads the 1-bit Armed Event Register using the :AER?
# query.
#
# The Armed Event Register bit goes low (0) when it is read using
# :AER? or when a *CLS command is issued.
# ====================================================================
def method_1():

    # Stop the oscilloscope.
    InfiniiVisionHD.query(":STOP;*OPC?")

    # Method 1: Initiate capture using :SINGle
    # ----------------------------------------------------------------
    print("Acquiring signal (Method 1, using :SINGle)...\n")
    now = time.perf_counter()

    # Clear all status registers before checking for new events.
    InfiniiVisionHD.write("*CLS")

    # Because the :AER? query will not work with :DIGitize (which is
    # blocking), use the :SINGle command to start the acquisition.
    InfiniiVisionHD.write(":SINGle")

    # Method 1: Determine if armed using :AER? query.
    # ----------------------------------------------------------------
    # Define armed criteria.
    ARMED = 1
```

```
    # Test for armed.
    ARMED_STATUS = int(InfiniiVisionHD.query(":AER?"))

    # Wait indefinitely until armed.
    while ARMED_STATUS != ARMED:
        # Check the status again after small delay.
        time.sleep(0.1)   # 100 ms delay to prevent excessive queries.
        ARMED_STATUS = int(InfiniiVisionHD.query(":AER?"))

    print("Oscilloscope is armed (method 1, using :AER? query)!")
    print("It took " + str(time.perf_counter() - now) +\
          " seconds to arm.\n")


# ================================================================
# Method 2: Read the Status Byte
# ----------------------------------------------------------------
# This method reads the Status Byte register's OPER bit (bit 7) using
# the "read status byte" function in VISA, which works during blocking
# commands and can therefore be used with the :DIGitize command.
#
# The Status Byte bits do NOT go low (0) when the register is read.
#
# The *CLS command will clear the Status Byte bits.
#
# CAUTION: The oscilloscope's status registers are not updated until
# the :DIGitize completes.  So, while the ARM may go true midway
# through the :DIGitize, it does not get reported to the status model
# until the :DIGitize completes, and therefore the STB does not
# reflect it as true until the :DIGitize completes.
# ================================================================
def method_2():

    # Stop the oscilloscope.
    InfiniiVisionHD.query(":STOP;*OPC?")

    # Method 2: Initiate capture using :DIGitize or :SINGle
    # ----------------------------------------------------------------
    print("Acquiring signal (Method 2, using :DIGitize)...\n")
    now = time.perf_counter()

    # Clear all status registers before checking for new events.
    InfiniiVisionHD.write("*CLS")

    # Mask out all bits in the Operation Status Event Register except
    # for the ARM bit.  "Unmask" only the arm bit.
    InfiniiVisionHD.write(":STATus:OPERation:ENABle 32")

    # Use the :DIGitize command to start the acquisition.
    InfiniiVisionHD.write(":DIGitize")

    # Method 2: Determine if armed by reading the Status Byte.
    # ----------------------------------------------------------------
    # Define register bit masks for the Status Byte Register
    ARM_BIT = 7
    # 1 leftshift 7 = 128 (bit 7 in the Status Byte Register)
```

```
    ARM_MASK = 1 << ARM_BIT

    # Define armed criteria.
    ARMED = 1 << ARM_BIT   # 1 leftshift 7 = 128

    # Test for armed.
    STATUS_BYTE = int(InfiniiVisionHD.read_stb())
    ARMED_STATUS = STATUS_BYTE & ARM_MASK
    # Note that you could also do:
    #  ARMED_STATUS = int(InfiniiVisionHD.query("*STB?))
    # BUT *STB? does not work with the blocking :DIGitize.

    # Wait indefinitely until armed.
    while ARMED_STATUS != ARMED:
        # Check the status again after small delay.
        time.sleep(0.1)   # 100 ms delay to prevent excessive queries.
        STATUS_BYTE = int(InfiniiVisionHD.read_stb())
        ARMED_STATUS = STATUS_BYTE & ARM_MASK

    print("Oscilloscope is armed (method 2, using Read STB function)!")
    print("It took " + str(time.perf_counter() - now) +\
          " seconds to arm.\n")


# ======================================================================
# Method 3: Query the Operation Status Event Register with
# :STATus:OPERation:EVENt?
# ----------------------------------------------------------------------
# This method reads the Operation Status Event Register's Wait Trig
# bit (bit 5) using the :STATus:OPERation:EVENt? query.
#
# The Operation Status Event Register bits are cleared (0) when the
# register is read.
#
# Also, the Wait Trig bit does NOT go low (0) when the oscilloscope
# becomes unarmed by starting or stopping another acquisition (before
# the first one finishes) or by changing the time scale.
#
# The Wait Trig bit is cleared by a *CLS command, by reading the
# Operation Status Event Register with the :STATus:OPERation:EVENt?
# query, or by reading the Armed Event Register regsiter with the
# :AER? query.
# ======================================================================
def method_3():

    # Stop the oscilloscope.
    InfiniiVisionHD.query(":STOP;*OPC?")

    # Method 3: Initiate capture using :SINGle
    # ------------------------------------------------------------------
    print("Acquiring signal (Method 3, using :SINGle)...\n")
    now = time.perf_counter()

    # Clear all status registers before checking for new events.
    InfiniiVisionHD.write("*CLS")

    # Because the :STATus:OPERation:EVENt? query will not work with
```

```
    # :DIGitize (which is blocking), use the :SINGle command to
    # start the acquisition.
    InfiniiVisionHD.write(":SINGle")

    # Method 3: Determine if armed using :STATus:OPERation:EVENt?
    # query.
    # ----------------------------------------------------------------
    # Define bit masks for the Operation Status Event Register
    ARM_BIT = 5
    # 1 leftshift 5 = 32 (bit 5 in the Operation Status Event
    # Register)
    ARM_MASK = 1 << ARM_BIT

    # Define armed criteria.
    ARMED = 1 << ARM_BIT   # 1 leftshift 5 = 32

    # Test for armed.
    STATUS_REGISTER = \
        int(InfiniiVisionHD.query(":STATus:OPERation:EVENt?"))
    ARMED_STATUS = STATUS_REGISTER & ARM_MASK

    # Wait indefinitely until armed.
    while ARMED_STATUS != ARMED:
       # Check the status again after small delay.
       time.sleep(0.1)   # 100 ms delay to prevent excessive queries.
       STATUS_REGISTER = \
        int(InfiniiVisionHD.query(":STATus:OPERation:EVENt?"))
       ARMED_STATUS = STATUS_REGISTER & ARM_MASK

    print("Oscilloscope is armed (method 3, using " +\
          ":STATus:OPERation:EVENt? query)!")
    print("It took " + str(time.perf_counter() - now) +\
          " seconds to arm.\n")


# ================================================================
# Main
# ================================================================

# Connect and initialize oscilloscope
# ----------------------------------------------------------------
# Define VISA Resource Manager & Install directory
rm = pyvisa.ResourceManager('C:\\Windows\\System32\\visa64.dll')

# Define and open the oscilloscope using the VISA address
InfiniiVisionHD = rm.open_resource(VISA_ADDRESS)

# Set the Global Timeout
InfiniiVisionHD.timeout = GLOBAL_TOUT

# Clear the instrument bus
InfiniiVisionHD.clear()

# Reset the oscilloscope.
InfiniiVisionHD.write("*RST")

# Autoscale to set up vertical scale and trigger level on Probe Comp.
```

```
InfiniiVisionHD.write(":CHANnel1:DISPlay OFF")
InfiniiVisionHD.write(":CHANnel2:DISPlay ON")
InfiniiVisionHD.write(":AUToscale:CHANnels DISPlayed")
InfiniiVisionHD.write(":AUToscale")

# Ensure a "long" time to arm (5 seconds) and not trigger immediately.
# ---------------------------------------------------------------------
# 10 second total capture, with trigger point in the middle = 5s to arm
InfiniiVisionHD.write(":TIMebase:RANGe 10")
# Prevent Auto trigger.
InfiniiVisionHD.write(":TRIGger:SWEep NORMal")

# Use the three methods to check whether the oscilloscope is armed.
# ---------------------------------------------------------------------
method_1()
method_2()
method_3()

# End of Script
# ---------------------------------------------------------------------
InfiniiVisionHD.clear()   # Clear communications interface
InfiniiVisionHD.close()   # Close communications interface
print("All done.")
```

Example: Waiting for IO Operation Complete

NOTE The IOC (IO Operation Complete) and IOF (IO Operation Failed) bits in the Operation Status Event Register identify when :WMEMory<r>:SAVE and other :SAVE and :RECall commands are completely done. To determine when a IO operation is complete, you should use these bits instead of the OPC (Operation Complete) bit in the Standard Event Status Register or the *OPC? query.

```
#!python3
# -*- coding: utf-8 -*-

# ******************************************************************
# This script using the Python language (http://www.python.org/) and
# the PyVISA package (http://pyvisa.readthedocs.org/) shows how to
# wait for IO operation completion in a Keysight InfiniiVision
# oscilloscope.
# ******************************************************************

# Import modules
# ----------------------------------------------------------------
import sys
import pyvisa
import time

# Initialization constants
# ----------------------------------------------------------------
# Get VISA address from Keysight IO Libraries Connection Expert
VISA_ADDRESS = "USB0::0x2A8D::0x4704::Unset::0::INSTR"
GLOBAL_TOUT =  20000   # IO timeout in milliseconds


# ================================================================
# Check for IO Operation Complete using :STATus:OPERation:EVENt? query.
# ----------------------------------------------------------------
# This method reads the Operation Status Register's IOC bit
# (bit 13) using the :STATus:OPERation:EVENt? query.
#
# The Operation Status Event Register bits are cleared (0) when the
# register is read.
#
# All status bits are cleared by a *CLS command.
# ================================================================
def wait_io_operation():

    # ------------------------------------------------------------
    now = time.perf_counter()
    # Define bit masks for the Operation Status Event Register
    IOC_BIT = 13
    # 1 leftshift 13 = 8192 (bit 13 in the Operation Status Event
    # Register)
    IOC_MASK = 1 << IOC_BIT

    # Define IO complete criteria.
```

```
    IO_COMPLETE = 1 << IOC_BIT   # 1 leftshift 13 = 8192

    # Test for armed.
    STATUS_REGISTER = \
        int(InfiniiVisionHD.query(":STATus:OPERation:EVENt?"))
    IO_COMPLETE_STATUS = STATUS_REGISTER & IOC_MASK

    # Wait indefinitely until armed.
    while IO_COMPLETE_STATUS != IO_COMPLETE:
       # Check the status again after small delay.
       time.sleep(0.1)   # 100 ms delay to prevent excessive queries.
       STATUS_REGISTER = \
        int(InfiniiVisionHD.query(":STATus:OPERation:EVENt?"))
       IO_COMPLETE_STATUS = STATUS_REGISTER & IOC_MASK

    time_in_seconds = str(time.perf_counter() - now)
    print("IO Operation Complete (from :STATus:OPERation:EVENt? query)."
)
    print("It took %s seconds for IO operation to complete.\n" % \
          time_in_seconds)


# ======================================================================
# Main
# ======================================================================

# Connect and initialize oscilloscope
# ----------------------------------------------------------------------
# Define VISA Resource Manager & Install directory
rm = pyvisa.ResourceManager('C:\\Windows\\System32\\visa64.dll')

# Define and open the oscilloscope using the VISA address
InfiniiVisionHD = rm.open_resource(VISA_ADDRESS)

# Set the Global Timeout
InfiniiVisionHD.timeout = GLOBAL_TOUT

# Clear the instrument bus
InfiniiVisionHD.clear()

# Reset the oscilloscope.
# InfiniiVisionHD.write("*RST")
# Or comment out to use the current oscilloscope setup.

# Autoscale to set up vertical scale and trigger level on channels 1 and
 2.
InfiniiVisionHD.write(":CHANnel1:DISPlay ON")
InfiniiVisionHD.write(":CHANnel2:DISPlay ON")
InfiniiVisionHD.write(":AUToscale:CHANnels DISPlayed")
InfiniiVisionHD.write(":AUToscale")

# Between :WMEMory<r>:SAVE commands, wait for IO complete.
# ----------------------------------------------------------------------
# Clear all status registers before checking for new events.
InfiniiVisionHD.write("*CLS")

InfiniiVisionHD.write(":WMEMory1:SAVE CHANnel1")
```

```
wait_io_operation()
InfiniiVisionHD.write(":WMEMory2:SAVE CHANnel2")
wait_io_operation()

# End of Script
# ------------------------------------------------------------------------
InfiniiVisionHD.clear()   # Clear communications interface
InfiniiVisionHD.close()   # Close communications interface
print("All done.")
```

42 Synchronizing Acquisitions

When remotely controlling an oscilloscope with programming commands, it is often necessary to know when the oscilloscope has finished the previous operation and is ready for the next command. The most common example is when an acquisition is started using the :DIGitize, :RUN, or :SINGle commands. Before a measurement result can be queried, the acquisition must complete. Too often fixed delays are used to accomplish this wait, but fixed delays often use excessive time or the time may not be long enough. A better solution is to use synchronous commands and status to know when the oscilloscope is ready for the next request.

Synchronization in the Programming Flow

Most remote programming follows these three general steps:

1. Set up the oscilloscope and device under test (see page 1270).
2. Acquire a waveform (see page 1270).
3. Retrieve results (see page 1270).

Set Up the Oscilloscope

Before making changes to the oscilloscope setup, it is best to make sure it is stopped using the :STOP command followed by the *OPC? query.

NOTE **It is not necessary to use *OPC?, hard coded waits, or status checking when setting up the oscilloscope. After the oscilloscope is configured, it is ready for an acquisition.**

Acquire a Waveform

When acquiring a waveform there are two possible methods used to wait for the acquisition to complete. These methods are blocking and polling. The table below details when each method should be chosen and why.

	Blocking Wait	Polling Wait
Use When	You know the oscilloscope *will* trigger based on the oscilloscope setup and device under test.	You know the oscilloscope *may or may not* trigger on the oscilloscope setup and device under test.
Advantages	No need for polling. Fastest method.	Remote interface will not timeout No need for device clear if no trigger.
Disadvantages	Remote interface may timeout. Device clear only way to get control of oscilloscope if there is no trigger.	Slower method. Requires polling loop. Requires known maximum wait time.
Implementation Details	See "Blocking Synchronization" on page 1271.	See "Polling Synchronization With Timeout" on page 1272.

Retrieve Results

Once the acquisition is complete, it is safe to retrieve measurements and statistics.

Blocking Synchronization

Use the :DIGitize command to start the acquisition. This blocks subsequent queries until the acquisition and processing is complete. For example:

```
'
' Synchronizing acquisition using blocking.
' ====================================================================

Option Explicit

Public myMgr As VisaComLib.ResourceManager
Public myScope As VisaComLib.FormattedIO488
Public varQueryResult As Variant
Public strQueryResult As String

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("USB0::0x2A8D::0x4704::Unset::0::INSTR")
  myScope.IO.Clear   ' Clear the interface.
  myScope.IO.Timeout = 10000   ' Set I/O communication timeout.

  ' Set up.
  ' --------------------------------------------------------------------
  myScope.WriteString ":TRIGger:MODE EDGE"
  myScope.WriteString ":TRIGger:EDGE:LEVel 2"
  myScope.WriteString ":TIMebase:SCALe 5e-8"

  ' Acquire.
  ' --------------------------------------------------------------------
  myScope.WriteString ":DIGitize"

  ' Get results.
  ' --------------------------------------------------------------------
  myScope.WriteString ":MEASure:RISetime"
  myScope.WriteString ":MEASure:RISetime?"
  varQueryResult = myScope.ReadNumber   ' Read risetime.
  Debug.Print "Risetime: " + _
      FormatNumber(varQueryResult * 1000000000, 1) + " ns"

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error:" + vbCrLf + Err.Description

End Sub
```

Polling Synchronization With Timeout

This example requires a timeout value so the operation can abort if an acquisition does not occur within the timeout period:

```
'
' Synchronizing acquisition using polling.
' ====================================================================

Option Explicit

Public myMgr As VisaComLib.ResourceManager
Public myScope As VisaComLib.FormattedIO488
Public varQueryResult As Variant
Public strQueryResult As String

#If VBA7 Then
    Private Declare PtrSafe Sub Sleep Lib "kernel32" (ByVal dwMillisecon
ds As LongPtr)
#Else
    Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Lo
ng)
#End If


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("USB0::0x2A8D::0x4704::Unset::0::INSTR")
  myScope.IO.Clear   ' Clear the interface.
  myScope.IO.Timeout = 60000   ' Set I/O communication timeout.

  ' Set up.
  ' --------------------------------------------------------------------
  ' Set up the trigger and horizontal scale.
  myScope.WriteString ":TRIGger:MODE EDGE"
  myScope.WriteString ":TRIGger:EDGE:LEVel 2"
  myScope.WriteString ":TIMebase:SCALe 5e-8"
  myScope.WriteString ":TRIGger:SWEep NORMal"

  ' Stop acquisitions and wait for the operation to complete.
  myScope.WriteString ":STOP"
  myScope.WriteString "*OPC?"
  strQueryResult = myScope.ReadString
  myScope.WriteString ":DISPlay:CLEar"

  ' Acquire.
  ' --------------------------------------------------------------------
  ' Start a single acquisition.
  myScope.WriteString ":SINGle"
```

```
  ' Oscilloscope is armed and ready, enable DUT here.
  Debug.Print "Oscilloscope is armed and ready, enable DUT."

  ' Look for RUN bit = stopped (acquisition complete).
  Dim lngTimeout As Long   ' Max millisecs to wait for single-shot.
  Dim lngElapsed As Long
  lngTimeout = 20000   ' 20 seconds.
  lngElapsed = 0

  Do While lngElapsed <= lngTimeout
    myScope.WriteString ":STATus:OPERation:CONDition?"
    varQueryResult = myScope.ReadNumber
    ' Mask RUN bit (bit 3, &H8).
    If (varQueryResult And &H8) = 0 Then
      Exit Do
    Else
      Sleep 100   ' Small wait to prevent excessive queries.
      lngElapsed = lngElapsed + 100
    End If
  Loop

  ' Get results.
  ' ------------------------------------------------------------------
  If lngElapsed < lngTimeout Then
    myScope.WriteString ":MEASure:RISetime"
    myScope.WriteString ":MEASure:RISetime?"
    varQueryResult = myScope.ReadNumber   ' Read risetime.
    Debug.Print "Risetime: " + _
        FormatNumber(varQueryResult * 1000000000, 1) + " ns"
  Else
    Debug.Print "Timeout waiting for single-shot trigger."
  End If

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error:" + vbCrLf + Err.Description

End Sub
```

Synchronizing with a Single-Shot Device Under Test (DUT)

The examples in "Blocking Synchronization" on page 1271 and "Polling Synchronization With Timeout" on page 1272 assume the DUT is continually running and therefore the oscilloscope will have more than one opportunity to trigger. With a single shot DUT, there is only one opportunity for the oscilloscope to trigger, so it is necessary for the oscilloscope to be armed and ready before the DUT is enabled.

NOTE **The blocking :DIGitize command cannot be used for a single shot DUT because once the :DIGitize command is issued, the oscilloscope is blocked from any further commands until the acquisition is complete.**

This example is the same "Polling Synchronization With Timeout" on page 1272 with the addition of checking for the armed event status.

```
'
' Synchronizing single-shot acquisition using polling.
' ====================================================================

Option Explicit

Public myMgr As VisaComLib.ResourceManager
Public myScope As VisaComLib.FormattedIO488
Public varQueryResult As Variant
Public strQueryResult As String

#If VBA7 Then
    Private Declare PtrSafe Sub Sleep Lib "kernel32" (ByVal dwMillisecon
ds As LongPtr)
#Else
    Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Lo
ng)
#End If


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("USB0::0x2A8D::0x4704::Unset::0::INSTR")
  myScope.IO.Clear   ' Clear the interface.
  myScope.IO.Timeout = 60000   ' Set I/O communication timeout.

  ' Set up.
  ' --------------------------------------------------------------------
  ' Set up the trigger and horizontal scale.
  myScope.WriteString ":TRIGger:MODE EDGE"
  myScope.WriteString ":TRIGger:EDGE:LEVel 2"
```

```
myScope.WriteString ":TIMebase:SCALe 5e-8"
myScope.WriteString ":TRIGger:SWEep NORMal"

' Stop acquisitions and wait for the operation to complete.
myScope.WriteString ":STOP"
myScope.WriteString "*OPC?"
strQueryResult = myScope.ReadString
myScope.WriteString ":DISPlay:CLEar"

' Acquire.
' ----------------------------------------------------------------
' Start a single acquisition.
myScope.WriteString ":SINGle"

' Wait until the trigger system is armed.
Do
  Sleep 100   ' Small wait to prevent excessive queries.
  myScope.WriteString ":AER?"
  varQueryResult = myScope.ReadNumber
Loop Until varQueryResult = 1

' Oscilloscope is armed and ready, enable DUT here.
Debug.Print "Oscilloscope is armed and ready, enable DUT."

' Now, look for RUN bit = stopped (acquisition complete).
Dim lngTimeout As Long   ' Max millisecs to wait for single-shot.
Dim lngElapsed As Long
lngTimeout = 20000   ' 20 seconds.
lngElapsed = 0

Do While lngElapsed <= lngTimeout
  myScope.WriteString ":STATus:OPERation:CONDition?"
  varQueryResult = myScope.ReadNumber
  ' Mask RUN bit (bit 3, &H8).
  If (varQueryResult And &H8) = 0 Then
    Exit Do
  Else
    Sleep 100   ' Small wait to prevent excessive queries.
    lngElapsed = lngElapsed + 100
  End If
Loop

' Get results.
' ----------------------------------------------------------------
If lngElapsed < lngTimeout Then
  myScope.WriteString ":MEASure:RISetime"
  myScope.WriteString ":MEASure:RISetime?"
  varQueryResult = myScope.ReadNumber   ' Read risetime.
  Debug.Print "Risetime: " + _
      FormatNumber(varQueryResult * 1000000000, 1) + " ns"
Else
  Debug.Print "Timeout waiting for single-shot trigger."
End If

Exit Sub

VisaComError:
```

```
    MsgBox "VISA COM Error:" + vbCrLf + Err.Description

End Sub
```

Synchronization with an Averaging Acquisition

When averaging, it is necessary to know when the average count has been reached. The :SINGle command does not average.

If it is known that a trigger will occur, a :DIGitize will acquire the complete number of averages, but if the number of averages is large, a timeout on the connection can occur.

The example below polls during the :DIGitize to prevent a timeout on the connection.

```
'
' Synchronizing in averaging acquisition mode.
' ====================================================================

Option Explicit

Public myMgr As VisaComLib.ResourceManager
Public myScope As VisaComLib.FormattedIO488
Public varQueryResult As Variant
Public strQueryResult As String

#If VBA7 Then
    Private Declare PtrSafe Sub Sleep Lib "kernel32" (ByVal dwMillisecon
ds As LongPtr)
#Else
    Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Lo
ng)
#End If


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("USB0::0x2A8D::0x4704::Unset::0::INSTR")
  myScope.IO.Clear   ' Clear the interface.
  myScope.IO.Timeout = 5000

  ' Set up.
  ' --------------------------------------------------------------------
  ' Set up the trigger and horizontal scale.
  myScope.WriteString ":TRIGger:SWEep NORMal"
  myScope.WriteString ":TRIGger:MODE EDGE"
  myScope.WriteString ":TRIGger:EDGE:LEVel 2"
  myScope.WriteString ":TIMebase:SCALe 5e-8"

  ' Stop acquisitions and wait for the operation to complete.
  myScope.WriteString ":STOP"
  myScope.WriteString "*OPC?"
  strQueryResult = myScope.ReadString
```

```
  ' Set up average acquisition mode.
  Dim lngAverages As Long
  lngAverages = 256
  myScope.WriteString ":ACQuire:COUNt " + CStr(lngAverages)
  myScope.WriteString ":ACQuire:TYPE AVERage"

  ' Save *ESE (Standard Event Status Enable register) mask
  ' (so it can be restored later).
  Dim varInitialESE As Variant
  myScope.WriteString "*ESE?"
  varInitialESE = myScope.ReadNumber

  ' Set *ESE mask to allow only OPC (Operation Complete) bit.
  myScope.WriteString "*ESE " + CStr(CInt("&H01"))

  ' Acquire using :DIGitize.  Set up OPC bit to be set when the
  ' operation is complete.
  ' ----------------------------------------------------------------
  myScope.WriteString ":DIGitize"
  myScope.WriteString "*OPC"

  ' Assume the oscilloscope will trigger, if not put a check here.

  ' Wait until OPC becomes true (bit 5 of Status Byte register, STB,
  ' from Standard Event Status register, ESR, is set).  STB can be
  ' read during :DIGitize without generating a timeout.
  Do
    Sleep 4000   ' Poll more often than the timeout setting.
    varQueryResult = myScope.IO.ReadSTB
  Loop While (varQueryResult And &H20) = 0

  ' Clear ESR and restore previously saved *ESE mask.
  myScope.WriteString "*ESR?"   ' Clear ESR by reading it.
  varQueryResult = myScope.ReadNumber
  myScope.WriteString "*ESE " + CStr(varInitialESE)

  ' Get results.
  ' ----------------------------------------------------------------
  myScope.WriteString ":WAVeform:COUNt?"
  varQueryResult = myScope.ReadNumber
  Debug.Print "Averaged waveforms: " + CStr(varQueryResult)

  myScope.WriteString ":MEASure:RISetime"
  myScope.WriteString ":MEASure:RISetime?"
  varQueryResult = myScope.ReadNumber   ' Read risetime.
  Debug.Print "Risetime: " + _
      FormatNumber(varQueryResult * 1000000000, 1) + " ns"

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error:" + vbCrLf + Err.Description

End Sub
```

Example: Blocking and Polling Synchronization

```
#!python3
# -*- coding: utf-8 -*-

# ******************************************************************
# This script using the Python language (http://www.python.org/) and
# the PyVISA package (http://pyvisa.readthedocs.org/) shows the two
# best synchronization methods for InfiniiVision oscilloscopes.
# Benefits and drawbacks of each method are described.  No error
# handling is provided except in the actual synchronization methods.
# ******************************************************************

# Import modules
# ----------------------------------------------------------------
import sys
import pyvisa
import time

# Initialization constants
# ----------------------------------------------------------------
# Get VISA address from Keysight IO Libraries Connection Expert
VISA_ADDRESS = \
    "USB0::0x2A8D::0x4704::Unset::0::INSTR"
GLOBAL_TOUT =  10000   # IO timeout in milliseconds

TIME_TO_TRIGGER = 10   # Time in seconds
# ----------------------------------------------------------------
# This is the time until the FIRST trigger event.
#
# While the script calculates a general timeout for the given setup,
# it cannot know when a trigger event will occur.  Thus, you must
# still set this value.
#
# This time is in addition to the calculated minimum timeout... so, if
# an oscilloscope might take say, 1 us to arm and acquire data, the
# signal might take 100 seconds before it occurs... this accounts for
# that.
#
# The SCOPE_ACQUISITION_TIME_OUT calculation pads this by 1.1.
# ----------------------------------------------------------------

TIME_BETWEEN_TRIGGERS = 0.025   # Time in seconds - for Average,
# Segmented, and Equivalent Time modes, else set to 0
# ----------------------------------------------------------------
# In Average, Segmented, and Equivalent Time modes, the oscilloscope
# makes repeated acquisitions.  This is similar to the above
# TIME_TO_TRIGGER, but it is the time BETWEEN triggers.  For example,
# it might take 10 seconds for the first trigger event, and then they
# might start occurring regularly at say, 1 ms intervals.  In that
# scenario, 15 seconds (a conservative number for 10s) would be good
# for TIME_TO_TRIGGER, and 2 ms (again conservative) would be good for
# TIME_BETWEEN_TRIGGERS.
#
# The default in this sample script is 0.025 seconds.  This is to make
```

```
# the sample work for the LINE trigger used in this script when the
# oscilloscope is in Average, Segmented, and Equivalent Time modes to
# force a trigger off of the AC input line (:TRIGger:EDGE:SOURce LINE)
# which runs at 50 or 60 Hz in most of the world (1/50 Hz -> 20 ms, so
# use 25 ms to be conservative).
#
# The SCOPE_ACQUISITION_TIME_OUT calculation pads this by 1.1.
# ----------------------------------------------------------------------


# ======================================================================
# Define a simple and fast function utilizing the blocking :DIGitize
# command in conjunction with *OPC?.
# ----------------------------------------------------------------------
#
# Benefits of this method:
#
#   -  Fastest, compact
#   -  Only way for Average Acquisition type:
#         -  The :SINGle command does not do a complete average.
#         -  Counting triggers with :RUN is much too slow.
#   -  Allows for easy synchronization with math functions
#   -  Don't have to deal with the status registers, which can be
#      confusing.
#   -  Potentially faster than polling_method(), for better throughput
#   -  Because it's faster, one can retrieve more accurate acquisition
#      times than with a polling method.
#   -  Works best for segmented memory if any post processing is done
#      on the oscilloscope, for example, measurements, lister, math,
#      as this does not come back until the processing is all done
#         -  In this scenario, :DIGitize does not reduce the sample
#            rate or memory depth.
#
# Drawbacks of this method:
#
#   -  Usually does not fill acquisition memory to the maximum
#      available, usually only on-screen data.
#
#   -  May not be at the highest sample rate (compared with the
#      polling_method).
#
#   -  Requires a well-chosen, hard-set timeout that will cover the
#      time to arm, trigger, and finish acquisition.
#
#   -  Requires Exception handling and a device_clear() for a
#      possible timeout (no trigger event).
#
#   -  Socket connection cannot do device_clear()
#
#   -  Because :DIGitize is a "specialized form of the :RUN" command,
#      on these oscilloscopes, that results in:
#
#         -  the sample rate MAY be reduced from using :SINGle -
#            usually at longer time scales - typically only acquires
#            what is on screen, though at the fastest time scales,
#            more than on screen data may be acquired.  Thus, for max
#            memory and max sample rate, use the polling_method(),
```

```
#            which uses :SINGle.
#
# How it works:
#
#   -  The :DIGitize command is a blocking command, and thus, all
#      other SCPI commands are blocked until :DIGitize is completely
#      done.  This includes any subsequent processing that is already
#      set up, such as math and measurements.
#
#      KEY POINT: However, :DIGitize does not prevent additional
#      commands from being sent to the queue or cause the remote
#      program to wait.  For example, if your program does something
#      like:
#
#         InfiniiVisionHD.write(":DIGitize")
#         print("Signal acquired.\n")
#
#      The "Signal acquired" message will be written immediately
#      after the :DIGitize is sent, not after the acquisition and
#      processing is complete.
#
#      To pause the program until the :DIGitize is complete, you must
#      wait for a query result after the :DIGitize.  For example, in
#      this case:
#
#         query_result = InfiniiVisionHD.query(":DIGitize;*OPC?")
#         print("Signal acquired.\n")
#
#      The "Signal acquired" message will be written after the
#      acquisition and processing is complete.  The *OPC? query is
#      appended to :DIGitize with a semi-colon (;), which
#      essentially ties it to the same thread in the parser. It is
#      immediately dealt with once :DIGitize finishes and gives a "1"
#      back to the program (whether the program uses it or not),
#      allowing the program to move on.
#
# Other Notes:
#
#   -  If you DO NOT know when a trigger will occur, you should set a
#      very long timeout.
#
#   -  The timeout should be adjusted before and after the :DIGitize
#      operation, though this is not absolutely required.
#
#   -  A :DIGitize can be aborted with a device clear, which also
#      stops the oscilloscope:
#      InfiniiVisionHD.clear()
#
#   -  :DIGItize disables the anti-aliasing feature (sample rate
#      dither) on all InfiniiVision and InfiniiVision X-Series
#      oscilloscopes.
#
#   -  :DIGitize temporarily blocks the front panel, and all front
#      panel presses are queued until :DIGitize is done.  So if you
#      change the vertical scale, it will not happen until the
#      acquisition is done.
#
```

```
#      The exception is that the Run/Stop button on the front panel
#      is NOT blocked (unless the front panel is otherwise locked by
#      ":SYSTem:LOCK ON").
# ======================================================================
def blocking_method():

    InfiniiVisionHD.timeout =  SCOPE_ACQUISITION_TIME_OUT
    # Time in milliseconds (PyVisa uses ms) to wait for the
    # oscilloscope to arm, trigger, finish acquisition, and finish
    # any processing.
    #
    # Note that this is a property of the device interface,
    # InfiniiVisionHD.
    #
# If doing repeated acquisitions, this should be done BEFORE the
    # loop, and changed again after the loop if the goal is to
    # achieve the best throughput.

    print("Acquiring signal(s)...")
    # Set up a try/except block to catch a possible timeout and exit.
    try:
        InfiniiVisionHD.query(":DIGitize;*OPC?")
        # Acquire the signal(s) with :DIGItize (blocking) and wait
        # until *OPC? comes back with a one.
        print("Signal acquired.")

        # Reset timeout back to what it was, GLOBAL_TOUT.
        InfiniiVisionHD.timeout =  GLOBAL_TOUT

    # Catch a possible timeout and exit.
    except Exception:
        print("The acquisition timed out, most likely due to no " \
              "trigger or improper setup causing no trigger. " \
              "Properly closing the oscilloscope connection and " \
              "exiting script.\n")
        InfiniiVisionHD.clear() # Clear communications interface;
                                # A device clear also aborts digitize.
        InfiniiVisionHD.close() # Close communications interface
        sys.exit("Exiting script.")


# ======================================================================
# Define a function using the non-blocking :SINGle command and polling
# on the Operation Status Condition Register
# ----------------------------------------------------------------------
#
# Benefits of this method:
#
#  -  Don't have to worry about interface timeouts.
#  -  Easy to expand to know when the oscilloscope is armed.
#  -  Don't have to worry about interface timeouts
#  -  Easy to expand to know when scope is armed, and triggered
#  -  MAY result in a higher sample rate than the blocking method
#  -  Always fills max available memory
#  -  Can use with a socket connection if desired
#
#
```

```
# Drawbacks of this method:
#
#  -  Slow, as you don't want to poll the oscilloscope too fast.
#
#  -  DOES NOT work for Equivalent time mode.  MUST use the blocking
#     method.
#
#  -  Slow
#
#  -  Does NOT work for Average Acquisition type
#       -  :SINGle does not do a complete average
#            -  It does a single acquisition as if it were in NORMal
#               acq. type
#            -  Counting triggers in :RUN is much too slow
#
#  -  Works for Segmented Memory, BUT if any post processing is done
#     on the oscilloscope, for example measurements, lister, math, as
#     this reports that the acquisition is done, which is correct,
#     BUT the processing is NOT done, and it will take an indefinite
#     amount of time to wait for that, though there is no way to tell
#     if it is done.  Use the blocking_method for Segmented Memory.
#
#  -  Can't be used effectively for synchronizing math functions
#
#     It can be done by applying an additional hard coded wait after
#     the acquisition is done.  At least 200 ms is suggested, more
#     may be required.
#
#     However, as long as the timeout is not excessively short, the
#     math happens fast enough that once :STATus:OPERation:CONDition?
#     comes back as done that one can just wait for it when it is
#     time to pull the math waveform.
#
#  -  Still need some maximum timeout (here MAX_TIME_TO_WAIT),
#     ideally, or the script will sit in the while loop forever if
#     there is no trigger event
#
#     Max timeout (here MAX_TIME_TO_WAIT) must also account for any
#     processing done (see comments on math above)
#
#     Max timeout (here MAX_TIME_TO_WAIT) must also account for time
#     to arm the scope and finish the acquisition
#
#     This arm/trigger/finish part is accounted for in the main script.
#
# How it works:
#
#  -  What really matters is the RUN bit in the Operation Condition
#     (not Event) Register.  This bit changes based on the
#     oscilloscope state.
#
#     If the oscilloscope is running, it is high (8), and low (0) if
#     it is stopped.
#
#     The only (best) way to get at this bit is with the
#     :STATus:OPERation:CONDition? query.  The Operation Condition Regis
ter
```

```
#     can reflect states for other oscilloscope properties, for
#     example, if the oscilloscope is armed, thus it can produce
#     values other than 0 (stopped) or 8 (running).
#
#     To handle that, the result of :STATus:OPERation:CONDition? is bitw
ise
#     ANDed (& in Python) with an 8.  This is called "unmasking".
#
#     Here, the "unmasking" is done in the script.  On the other
#     hand, it is possible to "mask" which bits get passed to the
#     summary bit to the next register below on the instrument
#     itself.  However, this method it typically only used when
#     working with the Status Byte, and not used here.
#
#  -  Why 8 = running = not done?
#
#     The Run bit is the 4th bit of the Operation Status Condition
#     (and Event) Registers.
#
#     The registers are binary and start counting at zero, thus the
#     4th bit is bit number 3, and 2^3 = 8, and thus it returns an
#     8 for high and a 0 for low.
#
#  -  Why the CONDITION and NOT the EVENT register?
#
#     The Condition register reflects the CURRENT state, while the
#     EVENT register reflects the first event that occurred since it
#     was cleared or read (as in: has it EVER happened?), thus the
#     CONDITION register is used.
#
#     Note that with this method using :SINGle, for InfiniiVision
#     X-Series oscilloscopes only, :SINGle itself forces the trigger
#     sweep mode into NORMal.  This does not happen with the blocking
#     method, using :DIGitize, or on the InfiniiVision notXs.
# ====================================================================
def polling_method():

    MAX_TIME_TO_WAIT = SCOPE_ACQUISITION_TIME_OUT/float(1000)
    # Time in seconds to wait for the oscilloscope to arm, trigger,
    # and finish acquisition.
    #
    # Note that this is NOT a property of the device interface,
    # InfiniiVisionHD, but rather some constant in the script to be
    # used later with the Python module "time", and will be used with
    # time.perf_counter().

    # Define "mask" bits.
    #
    # Mask condition for Run state in the Operation Status Condition
    # (and Event) Register.
    #
    # Refer to Programmer's Guide chapters on Status Reporting and
    # Synchronizing Acquisitions.
    RUN_BIT = 3
    # The run bit is the 4th bit (see next set of comments @
    # Completion Criteria).
    RUN_MASK = 1<<RUN_BIT
```

```
    # This basically means:  2^3 = 8, or rather, in Python 2**3
    # (<< is a left shift; left shift is fastest); this is used later
    # to "unmask" the result of the Operation Status Event Register
    # as there is no direct access to the RUN bit.

    # Define completion criterion:
    ACQ_DONE = 0
    # Means the oscilloscope is stopped
    ACQ_NOT_DONE = 1<<RUN_BIT
    # Means the oscilloscope is running; value is 8.  This is the
    # 4th bit of the Operation Status Condition (and Event) Register.
    # The registers are binary and start counting at zero, thus the
    # 4th bit (4th position in a binary representation of decimal
    # 8 = 2^3 = (1 left shift 3).  This is either High (running = 8)
    # or low (stopped and therefore done with acquisition = 0).

    print("Acquiring signal(s)...")

    # Define acquisition start time.  This is in seconds.
    StartTime = time.perf_counter()

    # Begin Acquisition with *CLS and the non-blocking :SINGle
    # command, concatenated together. The *CLS clears all (non-mask)
    # registers & sets them to 0;
    InfiniiVisionHD.write("*CLS;:SINGle")

    # Initialize the loop entry condition (assume Acq. is not done).
    Acq_State = ACQ_NOT_DONE

    # Poll the oscilloscope until Acq_State is a one. (This is NOT a
    # "Serial Poll.")
    while Acq_State == ACQ_NOT_DONE and (time.perf_counter() - StartTime
                                         <= MAX_TIME_TO_WAIT):

        # Ask oscilloscope if it is done with the acquisition via the
        # Operation Status Condition (not Event) Register.
        # The Condition register reflects the CURRENT state, while
        # the EVENT register reflects the first event that occurred
        # since it was cleared or read, thus the CONDITION register
        # is used.
        #
        # DO NOT do:
        # InfiniiVisionHD.query("*CLS;SINGle;STATus:OPERation:CONDition?
")
        # as putting :STATus:OPERation:CONDition? right after :SINgle
        # doesn't work reliably
        #
        # The oscilloscope SHOULD trigger, but it sits there with the
        # Single hard key on the oscilloscope lit yellow; pressing
        # this key causes a trigger.
        Status = int(InfiniiVisionHD.query(":STATus:OPERation:CONDition?
"))

        # Bitwise AND of the Status and RUN_MASK.  This exposes ONLY
        # the 4th bit, which is either High (running = 8) or low
        # (stopped and therefore done with acquisition = 0)
        Acq_State = (Status & RUN_MASK)
```

```
            if Acq_State == ACQ_DONE:
                break # Break out of while loop so that the 100 ms pause
                # below is not incurred if done.
            time.sleep(0.1)   # Pause 100 ms to prevent excessive queries
            # This can actually be set a little faster, at 0.045.  The
            # point here is that:
            #
            #   1.  If there are other things being controlled, going too
            #       fast can tie up the bus.
            #   2.  Going faster does not work on all scopes.  The
            #       symptom of this not working is:
            #
            # The oscilloscope SHOULD trigger, but it sits there with the
            # Single hard key on the oscilloscope lit yellow; pressing
            # this key causes a trigger.
            #
            # The pause should be at the end of the loop, so that the
            # oscilloscope is immediately asked if it is done.
            #
            # Loop exits when Acq_State != NOT_DONE, that is, it exits
            # the loop when it is DONE or if the max wait time is
            # exceeded.

        if Acq_State == ACQ_DONE: # Acquisition fully completed
            print("Signal acquired.")
        else: # Acquisition failed for some reason
            print("Max wait time exceeded.")
            print("This happens if there was no trigger event.")
            print("Adjust settings accordingly.\n")
            print("Properly closing oscilloscope connection and exiting " \
                  "script.\n")
            InfiniiVisionHD.clear() # Clear communications interface
            InfiniiVisionHD.query(":STOP;*OPC?")
            # Stop the oscilloscope
            InfiniiVisionHD.close() # Close communications interface
            sys.exit("Exiting script.")


# ==========================================================================
# Do Something with data... save, export, additional analysis...
# ==========================================================================
def do_something_with_data():

    # For example, make a peak-peak voltage measurement on channel 1:
    Vpp_Ch1 = \
        str(InfiniiVisionHD.query("MEASure:VPP? CHANnel1")).strip("\n")
    # The result has a newline, so remove it with .strip("\n")
    print("Vpp Ch1 = " + Vpp_Ch1 + " V\n")


# ==========================================================================
# Main code
# ==========================================================================

# Connect and initialize oscilloscope
# --------------------------------------------------------------------------
# Define VISA Resource Manager & Install directory
```

```
rm = pyvisa.ResourceManager('C:\\Windows\\System32\\visa64.dll')

# Define and open the oscilloscope using the VISA address
try:
    InfiniiVisionHD = rm.open_resource(VISA_ADDRESS)
except Exception:
    print("Unable to connect to oscilloscope at " + str(VISA_ADDRESS)\
          + ". Aborting script.\n")
    sys.exit()

# Set the Global Timeout
InfiniiVisionHD.timeout = GLOBAL_TOUT

# Clear the instrument bus
InfiniiVisionHD.clear()

# Clear all status registers and errors
InfiniiVisionHD.write("*CLS")

try:

    # Set up the oscilloscope
    # ----------------------------------------------------------------
    # Note that you would normally perform a reset (default setup) if
    # you were to create the setup from scratch... But here we will
    # use the oscilloscope "as is" for the most part.
    # InfiniiVisionHD.query("*RST;*OPC?")   # Resets the oscilloscope

    # Always stop the oscilloscope when making any changes.
    InfiniiVisionHD.query(":STOP;*OPC?")

    # Always use normal trigger sweep, never auto.
    InfiniiVisionHD.write(":TRIGger:SWEep NORMal")

    # For this example, the oscilloscope will trigger on CHANnel1.
    # You can use the Probe Comp input signal.
    InfiniiVisionHD.query(":TRIGger:EDGE:SOURce CHANnel1;*OPC?")

    # Clear the display (so you can see the waveform being acquired -
    # otherwise, there is no need for this).
    InfiniiVisionHD.write(":DISPlay:CLEar")

    # Calculate acquisition timeout/wait time by short, overestimate
    # method
    # ----------------------------------------------------------------

    # Create some default variables
    N_AVERAGES = 1
    N_SEGMENTS = 1

    # Get some info about the scope time base setup
    HO         = float(InfiniiVisionHD.query(":TRIGger:HOLDoff?"))
    T_RANGE    = float(InfiniiVisionHD.query(":TIMebase:RANGe?"))
    T_POSITION = float(InfiniiVisionHD.query(":TIMebase:POSition?"))

    # Determine Acquisition Type and Mode:
    ACQ_TYPE = str(InfiniiVisionHD.query(":ACQuire:TYPE?").strip("\n"))
```

```
    ACQ_MODE = str(InfiniiVisionHD.query(":ACQuire:MODE?").strip("\n"))

    if ACQ_MODE == "SEGM":
        N_SEGMENTS = \
            float(InfiniiVisionHD.query(":ACQuire:SEGMented:COUNt?"))
        # Note that if there are a lot of analysis associated segments,
        # for example, serial data decode, the timeout will likely need
        # to be longer than calculated.
        #
        # You are encouraged to manually set up the oscilloscope in
        # this case, as it will be used, time it, and use that, with
        # a little overhead.
        #
        # Blocking method is recommended for Segmented Memory mode.
    elif ACQ_TYPE == "AVER":
        N_AVERAGES = float(InfiniiVisionHD.query(":ACQuire:COUNt?"))

    # Calculate acuisition timeout by overestimate method:
    # Recall that PyVisa timeouts are in ms, so multiply by 1000.
    SCOPE_ACQUISITION_TIME_OUT = (float(TIME_TO_TRIGGER)*1.1 +
        (T_RANGE*2.0 + abs(T_POSITION)*2.0 + HO*1.1 +
        float(TIME_BETWEEN_TRIGGERS)*1.1)*N_SEGMENTS*N_AVERAGES)*1000.0

    ## Ensure the timeout is no less than 10 seconds
    if SCOPE_ACQUISITION_TIME_OUT < 10000.0:
        SCOPE_ACQUISITION_TIME_OUT = 10000.0

    # Acquire Signal
    # ----------------------------------------------------------------
    # Choose blocking_method or polling_method.  These were defined as
    # functions in case you want to use them repeatedly.
    blocking_method()
    # If Acquisition Type is Average, always use blocking_method() to
    # get complete average.
    do_something_with_data()

    polling_method()
    do_something_with_data()

    # Done - cleanup
    InfiniiVisionHD.clear() # Clear communications interface
    InfiniiVisionHD.close() # Close communications interface
except KeyboardInterrupt:
    InfiniiVisionHD.clear()
    InfiniiVisionHD.query(":STOP;*OPC?")
    InfiniiVisionHD.write(":SYSTem:LOCK OFF")
    InfiniiVisionHD.clear()
    InfiniiVisionHD.close()
    sys.exit("User Interupt.  Properly closing oscilloscope and "
             "aborting script.")
except Exception:
    InfiniiVisionHD.clear()
    InfiniiVisionHD.query(":STOP;*OPC?")
    InfiniiVisionHD.write(":SYSTem:LOCK OFF")
    InfiniiVisionHD.clear()
    InfiniiVisionHD.close()
    sys.exit("Something went wrong.  Properly closing oscilloscope "
```

```
            "and aborting script.")

# End of Script
# ------------------------------------------------------------------------
print("Done.")
```

43 More About Oscilloscope Commands

Command Classifications

To help you use existing programs with your oscilloscope, or use current programs with the next generation of Keysight InfiniiVision oscilloscopes, commands are classified by the following categories:

- "Core Commands" on page 1292
- "Non-Core Commands" on page 1292
- "Obsolete Commands" on page 1292

Core Commands

Core commands are a common set of commands that provide basic oscilloscope functionality on this oscilloscope and future Keysight InfiniiVision 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.

Non-Core Commands

Non-core commands are commands that provide specific features, but are not universal across all Keysight InfiniiVision oscilloscope models. Non-core commands may be modified or deleted in the future. With a command structure as complex as the one for your oscilloscope, some evolution over time is inevitable. Keysight's intent is to continue to expand command subsystems, such as the rich and evolving trigger feature set.

Obsolete Commands

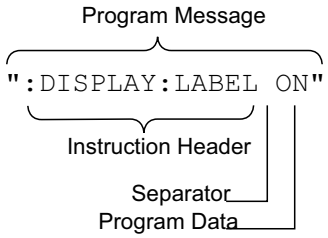
Obsolete commands are older forms of commands that are provided to reduce customer rework for existing systems and programs. Generally, these commands are mapped onto some of the Core and Non-core commands, but may not strictly have the same behavior as the new command. None of the obsolete commands 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:

- Chapter 39, “Obsolete and Discontinued Commands,” starting on page 1233

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" on page 1294), 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.

":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" on page 1295
- "Compound Command Headers" on page 1295
- "Common Command Headers" on page 1295

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" on page 1296 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 Units Within a program message, there can be multiple *program message units* separated by semicolons (;). Each program message unit represents a separate command or query. See "Multiple Program Message Units" on page 1297.

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 programming 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.

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.

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>
```

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

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 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 or zoomed (delayed). The character program data in this case may be MAIN or WINDow. The command :TIMebase:MODE WINDow sets the time base mode to zoomed.

The available mnemonics for character program data are always included with the command'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.

Multiple Program Message Units

Within a program message, there can be multiple program message units separated by semicolons (;). Each program message unit represents a separate command or query. The program message syntax is:

```
<program message unit>;<program message unit><terminator>
```

For example, in the program message ":TIMebase:REFerence CENTer;:DISPlay:LABel ON", one program message unit is ":TIMebase:REFerence CENTer" and the other program message unit is ":DISPlay:LABel ON".

Some commands and queries are not supported in program messages that have multiple program message units (for various reasons). Sending these commands and queries within multiple program message units will give unexpected errors.

Commands Not Supported in Multiple Program Message Units

These commands are not supported in program messages that have multiple program message units:

- ":BUS<n>:BITS" on page 252
- ":BUS<n>:MASK" on page 257
- ":MEASure:FFT:THD" on page 562
- ":MEASure:TEDGe" on page 608
- ":RECall:ARBitrary[:STARt]" on page 678
- ":RECall:DBC[:STARt]" on page 679
- ":RECall:LDF[:STARt]" on page 681
- ":RECall:WMEMory<r>[:STARt]" on page 685
- ":SAVE:ARBitrary[:STARt]" on page 691
- ":SBUS<n>:IIC:TRIGger:PATTern:DATA" on page 758
- ":SBUS<n>:IIC:TRIGger:PATTern:DATa2" on page 759
- ":SBUS<n>:UART:FRAMing" on page 808
- ":TRIGger:PATTern" on page 1112
- ":WGEN<w>:ARBitrary:DATA" on page 1187

Queries Not Supported in Multiple Program Message Units

These queries are not supported in program messages that have multiple program message units:

- :BUS<n>:BITS? (see page 252)
- :CHANnel<n>:PROBe:ID? (see page 287)
- :MEASure:FFT:THD? (see page 562)

- ":MEASure:RESults?" on page 590
- :MEASure:TEDGe? (see page 609)
- ":SERial?" on page 222
- ":SYSTem:DIDentifier?" on page 1026
- ":SYSTem:HID?" on page 1035

See Also
- "Tree Traversal Rules and Multiple Commands" on page 1298

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.

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 Keysight 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><ter
minator>
```

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:LABel ON"
```

NOTE

The leading colon before DISPlay:LABel ON tells the parser to go back to the root of the command tree. The parser can then see the DISPlay:LABel ON command. The space between REFerence and CENter is required; so is the space between LABel and ON.

Multiple commands may be any combination of compound and simple 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 Keysight 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.

44 Programming Examples

The example programs in this manual are ASCII text files that can be cut from the help file and pasted into your favorite text editor.

See Also

- You can find additional programming examples for the InfiniiVision HD3-Series oscilloscopes on the Keysight Technologies website at: www.keysight.com/find/HD3-Series-examples

VISA COM Examples

VISA COM Example in Visual Basic

To run this example in Visual Basic for Applications (VBA):

1. Start the application that provides Visual Basic for Applications (for example, Microsoft Excel).
2. Press ALT+F11 to launch the Visual Basic editor.
3. Reference the Keysight VISA COM library:
   a. Choose **Tools > References...** from the main menu.
   b. In the References dialog, check the "VISA COM 5.14 Type Library".
   c. Click **OK**.
4. Choose **Insert > Module**.
5. Cut-and-paste the code that follows into the editor.
6. Edit the program to use the VISA address of your oscilloscope, and save the changes.
7. Run the program.

```
'
' Keysight VISA COM Example in Visual Basic
' ----------------------------------------------------------------------
' This program illustrates a few commonly-used programming
' features of your Keysight oscilloscope.
' ----------------------------------------------------------------------

Option Explicit

Public myMgr As VisaComLib.ResourceManager
Public myScope As VisaComLib.FormattedIO488
Public varQueryResult As Variant
Public strQueryResult As String

' For Sleep subroutine.
#If VBA7 Then
    Private Declare PtrSafe Sub Sleep Lib "kernel32" (ByVal dwMillisecon
ds As LongPtr)
#Else
    Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Lo
ng)
#End If
```

```
'
' Main Program
' ---------------------------------------------------------------------

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("USB0::0x2A8D::0x4704::Unset::0::INSTR")
  myScope.IO.Clear   ' Clear the interface.
  myScope.IO.Timeout = 20000   ' Set I/O communication timeout.

  ' Initialize - start from a known state.
  Initialize

  ' Capture data.
  Capture

  ' Analyze the captured waveform.
  Analyze

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error:" + vbCrLf + Err.Description
  End

End Sub

'
' Initialize the oscilloscope to a known state.
' ---------------------------------------------------------------------

Private Sub Initialize()

  On Error GoTo VisaComError

  ' Get and display the device's *IDN? string.
  strQueryResult = DoQueryString("*IDN?")
  Debug.Print "Identification string: " + strQueryResult

  ' Clear status and load the default setup.
  DoCommand "*CLS"
  DoCommand "*RST"

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error:" + vbCrLf + Err.Description
  End

End Sub
```

```
'
' Capture the waveform.
' -----------------------------------------------------------------------

Private Sub Capture()

  On Error GoTo VisaComError

  ' Use auto-scale to automatically configure oscilloscope.
  ' -----------------------------------------------------------------------
  DoCommand ":AUToscale"

  ' Set trigger mode (EDGE, PULSe, PATTern, etc., and input source.
  DoCommand ":TRIGger:MODE EDGE"
  Debug.Print "Trigger mode: " + _
      DoQueryString(":TRIGger:MODE?")

  ' Set EDGE trigger parameters.
  DoCommand ":TRIGger:EDGE:SOURce CHANnel1"
  Debug.Print "Trigger edge source: " + _
      DoQueryString(":TRIGger:EDGE:SOURce?")

  DoCommand ":TRIGger:EDGE:LEVel 1.5"
  Debug.Print "Trigger edge level: " + _
      DoQueryString(":TRIGger:EDGE:LEVel?")

  DoCommand ":TRIGger:EDGE:SLOPe POSitive"
  Debug.Print "Trigger edge slope: " + _
      DoQueryString(":TRIGger:EDGE:SLOPe?")

  ' Save oscilloscope configuration.
  ' -----------------------------------------------------------------------
  Dim byteSetupSave() As Byte
  byteSetupSave = DoQueryIEEEBlock_UI1(":SYSTem:SETup?")

  ' Output file name:
  Dim strOutPath As String
  strOutPath = "c:\scope\config\setup.scp"

  ' If file exists, delete it.
  If (Dir(strOutPath) <> "") Then
    Kill strOutPath
  End If

  ' Output setup string to a file:
  Dim hFileOut As Integer
  hFileOut = FreeFile
  Open strOutPath For Binary Access Write Lock Write As hFileOut
  Put hFileOut, , byteSetupSave   ' Write data.
  Close hFileOut   ' Close file.

  ' Display number of bytes saved.
  Dim nOutLen As Long
  nOutLen = UBound(byteSetupSave) - LBound(byteSetupSave) + 1
  Debug.Print "Setup bytes saved: " + CStr(nOutLen)

  ' Change settings with individual commands:
```

```
    ' ------------------------------------------------------------------

    ' Set vertical scale and offset.
    DoCommand ":CHANnel1:SCALe 0.05"
    Debug.Print "Channel 1 vertical scale: " + _
        DoQueryString(":CHANnel1:SCALe?")

    DoCommand ":CHANnel1:OFFSet -1.5"
    Debug.Print "Channel 1 vertical offset: " + _
        DoQueryString(":CHANnel1:OFFSet?")

    ' Set horizontal scale and offset.
    DoCommand ":TIMebase:SCALe 0.0002"
    Debug.Print "Timebase scale: " + _
        DoQueryString(":TIMebase:SCALe?")

    DoCommand ":TIMebase:POSition 0.0"
    Debug.Print "Timebase position: " + _
        DoQueryString(":TIMebase:POSition?")

    ' Set the acquisition type (NORMal, PEAK, or AVERage).
    DoCommand ":ACQuire:TYPE NORMal"
    Debug.Print "Acquire type: " + _
        DoQueryString(":ACQuire:TYPE?")

    ' Or, configure by loading a previously saved setup.
    ' ------------------------------------------------------------------
    Dim byteSetupRestore() As Byte

    ' Output file name:
    Dim strInPath As String
    strInPath = "c:\scope\config\setup.scp"

    ' Get setup string from file:
    Dim hFileIn As Integer
    hFileIn = FreeFile
    Open strInPath For Binary Access Read As hFileIn
    ReDim byteSetupRestore(0 To LOF(hFileIn) - 1)
    Get hFileIn, , byteSetupRestore   ' Read data.
    Close hFileIn   ' Close file.

    ' Write setup string back to oscilloscope using ":SYSTem:SETup"
    ' command:
    DoCommandIEEEBlock ":SYSTem:SETup", byteSetupRestore

    ' Display number of bytes restored.
    Dim nInLen As Long
    nInLen = UBound(byteSetupRestore) - LBound(byteSetupRestore) + 1
    Debug.Print "Setup bytes restored: " + CStr(nInLen)

    ' Capture an acquisition using :DIGitize.
    ' ------------------------------------------------------------------
    DoCommand ":DIGitize CHANnel1"

    Exit Sub

VisaComError:
```

```
  MsgBox "VISA COM Error:" + vbCrLf + Err.Description
  End

End Sub

'
' Analyze the captured waveform.
' --------------------------------------------------------------------

Private Sub Analyze()

  On Error GoTo VisaComError

  ' Make a couple of measurements.
  ' --------------------------------------------------------------------
  DoCommand ":MEASure:SOURce CHANnel1"
  Debug.Print "Measure source: " + _
      DoQueryString(":MEASure:SOURce?")

  DoCommand ":MEASure:FREQuency"
  varQueryResult = DoQueryNumber(":MEASure:FREQuency?")
  MsgBox "Frequency:" + vbCrLf + _
      FormatNumber(varQueryResult / 1000, 4) + " kHz"

  DoCommand ":MEASure:VAMPlitude"
  varQueryResult = DoQueryNumber(":MEASure:VAMPlitude?")
  MsgBox "Vertical amplitude:" + vbCrLf + _
      FormatNumber(varQueryResult, 4) + " V"


  ' Download the screen image.
  ' --------------------------------------------------------------------
  ' Get screen image.
  Dim byteScreen() As Byte
  byteScreen = DoQueryIEEEBlock_UI1(":DISPlay:DATA? PNG")

  ' Save screen image to a file.
  Dim strPath As String
  strPath = "c:\scope\data\screen.png"
  If Len(Dir(strPath)) Then
    Kill strPath   ' Remove file if it exists.
  End If

  Dim hFile As Long
  hFile = FreeFile
  Open strPath For Binary Access Write Lock Write As hFile
  Put hFile, , byteScreen   ' Write data.
  Close hFile   ' Close file.
  MsgBox "Screen image (" + CStr(UBound(byteScreen) + 1) + _
      " bytes) written to " + strPath


  ' Download waveform data.
  ' --------------------------------------------------------------------

  ' Set the waveform points mode.
  DoCommand ":WAVeform:POINts:MODE RAW"
```

```
Debug.Print "Waveform points mode: " + _
    DoQueryString(":WAVeform:POINts:MODE?")

' Get the number of waveform points available.
DoCommand ":WAVeform:POINts 10240"
Debug.Print "Waveform points available: " + _
    DoQueryString(":WAVeform:POINts?")

' Set the waveform source.
DoCommand ":WAVeform:SOURce CHANnel1"
Debug.Print "Waveform source: " + _
    DoQueryString(":WAVeform:SOURce?")

' Choose the format of the data returned (WORD, BYTE, ASCII):
DoCommand ":WAVeform:FORMat WORD"
Debug.Print "Waveform format: " + _
    DoQueryString(":WAVeform:FORMat?")

' Specify the byte order in the word format data:
DoCommand ":WAVeform:BYTeorder LSBF"
Debug.Print "Waveform byte order: " + _
    DoQueryString(":WAVeform:BYTeorder?")

' Display the waveform settings:
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

Preamble() = DoQueryNumbers(":WAVeform:PREamble?")

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)

If intFormat = 0 Then
  Debug.Print "Waveform format: BYTE"
ElseIf intFormat = 1 Then
  Debug.Print "Waveform format: WORD"
ElseIf intFormat = 4 Then
  Debug.Print "Waveform format: ASCii"
End If
```

```
If intType = 0 Then
  Debug.Print "Acquisition type: NORMal"
ElseIf intType = 1 Then
  Debug.Print "Acquisition type: PEAK"
ElseIf intType = 2 Then
  Debug.Print "Acquisition type: AVERage"
ElseIf intType = 3 Then
  Debug.Print "Acquisition type: HRESolution"
End If

Debug.Print "Waveform points: " + _
    FormatNumber(lngPoints, 0)

Debug.Print "Waveform average count: " + _
    FormatNumber(lngCount, 0)

Debug.Print "Waveform X increment: " + _
    Format(dblXIncrement, "Scientific")

Debug.Print "Waveform X origin: " + _
    Format(dblXOrigin, "Scientific")

Debug.Print "Waveform X reference: " + _
    FormatNumber(lngXReference, 0)

Debug.Print "Waveform Y increment: " + _
    Format(sngYIncrement, "Scientific")

Debug.Print "Waveform Y origin: " + _
    FormatNumber(sngYOrigin, 0)

Debug.Print "Waveform Y reference: " + _
    FormatNumber(lngYReference, 0)

' Get the waveform data
varQueryResult = DoQueryIEEEBlock_UI1(":WAVeform:DATA?")
Debug.Print "Number of data values: " + _
    CStr((UBound(varQueryResult) + 1) / 2)

' Set up output file:
strPath = "c:\scope\data\waveform_data.csv"

' Open file for output.
Open strPath For Output Access Write Lock Write As hFile

' Output waveform data in CSV format.
Dim lngDataValue As Long
Dim lngI As Long

For lngI = 0 To UBound(varQueryResult) Step 2
  lngDataValue = varQueryResult(lngI) + _
      (varQueryResult(lngI + 1) * 256)

  ' Write time value, voltage value.
  Print #hFile, _
      FormatNumber(dblXOrigin + ((lngI / 2) * dblXIncrement), 9) + _
      ", " + _
```

```
        FormatNumber(((lngDataValue - lngYReference) * _
        sngYIncrement) + sngYOrigin)

  Next lngI

  ' Close output file.
  Close hFile   ' Close file.
  MsgBox "Waveform format WORD data written to " + _
      "c:\scope\data\waveform_data.csv."

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error:" + vbCrLf + Err.Description
  End

End Sub

Private Sub DoCommand(command As String)

  On Error GoTo VisaComError

  myScope.WriteString command
  CheckInstrumentErrors

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error: " + vbCrLf + CStr(Err.Number) + ", " + _
      Err.Source + ", " + _
      Err.Description, vbExclamation, "VISA COM Error"
  End

End Sub

Private Sub DoCommandIEEEBlock(command As String, data As Variant)

  On Error GoTo VisaComError

  Dim strErrors As String

  myScope.WriteIEEEBlock command, data
  CheckInstrumentErrors

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error: " + vbCrLf + CStr(Err.Number) + ", " + _
      Err.Source + ", " + _
      Err.Description, vbExclamation, "VISA COM Error"
  End

End Sub

Private Function DoQueryString(query As String) As String

  On Error GoTo VisaComError
```

```
  myScope.WriteString query
  DoQueryString = myScope.ReadString
  CheckInstrumentErrors

  Exit Function

VisaComError:
  MsgBox "VISA COM Error: " + vbCrLf + CStr(Err.Number) + ", " + _
      Err.Source + ", " + _
      Err.Description, vbExclamation, "VISA COM Error"
  End

End Function

Private Function DoQueryNumber(query As String) As Variant

  On Error GoTo VisaComError

  myScope.WriteString query
  DoQueryNumber = myScope.ReadNumber
  CheckInstrumentErrors

  Exit Function

VisaComError:
  MsgBox "VISA COM Error: " + vbCrLf + CStr(Err.Number) + ", " + _
      Err.Source + ", " + _
      Err.Description, vbExclamation, "VISA COM Error"
  End

End Function

Private Function DoQueryNumbers(query As String) As Variant()

  On Error GoTo VisaComError

  Dim strErrors As String

  myScope.WriteString query
  DoQueryNumbers = myScope.ReadList
  CheckInstrumentErrors

  Exit Function

VisaComError:
  MsgBox "VISA COM Error: " + vbCrLf + CStr(Err.Number) + ", " + _
      Err.Source + ", " + _
      Err.Description, vbExclamation, "VISA COM Error"
  End

End Function

Private Function DoQueryIEEEBlock_UI1(query As String) As Variant

  On Error GoTo VisaComError
```

```
  myScope.WriteString query
  DoQueryIEEEBlock_UI1 = myScope.ReadIEEEBlock(BinaryType_UI1)
  CheckInstrumentErrors

  Exit Function

VisaComError:
  MsgBox "VISA COM Error: " + vbCrLf + CStr(Err.Number) + ", " + _
      Err.Source + ", " + _
      Err.Description, vbExclamation, "VISA COM Error"
  End

End Function

Private Sub CheckInstrumentErrors()

  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
```

VISA COM Example in C#

To compile and run this example in Microsoft Visual Studio 2019:

1. Open Visual Studio.
2. Create a new Visual C#, Windows, Console Application project using the **.NET Framework 4.7.2**.
3. Cut-and-paste the code that follows into the C# source file.
4. Edit the program to use the VISA address of your oscilloscope.

5 Add a reference to the VISA COM 5.14 Type Library:

   a Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment.

   b Choose **Add > Reference...**.

   c In the Reference Manager dialog box, select the **COM** hierarchy on the left.

   d Select **VISA COM 5.14 Type Library**; then click **OK**.

6 In the Solution Explorer, expand References, and select **VisaComLib**. Then, in the Properties window, change **Embed Interop Types** to **False**.

7 Build and run the program.

For more information, see the VISA COM Help that comes with Keysight IO Libraries Suite.

```
/*
 * Keysight VISA COM Example in C#
 * ----------------------------------------------------------------------
 * This program illustrates a few commonly used programming
 * features of your Keysight oscilloscope.
 * ----------------------------------------------------------------------
 */

using System;
using System.IO;
using System.Text;
using Ivi.Visa.Interop;
using System.Runtime.InteropServices;

namespace InfiniiVision
{
  class VisaComInstrumentApp
  {
    private static VisaComInstrument myScope;

    public static void Main(string[] args)
    {
      try
      {
        myScope = new
          VisaComInstrument("USB0::0x2A8D::0x4704::Unset::0::INSTR");
        myScope.SetTimeoutSeconds(10);

        // Initialize - start from a known state.
        Initialize();

        // Capture data.
        Capture();

        // Analyze the captured waveform.
        Analyze();
      }
      catch (System.ApplicationException err)
      {
```

```
    Console.WriteLine("*** VISA COM Error : " + err.Message);
  }
  catch (System.SystemException err)
  {
    Console.WriteLine("*** System Error Message : " + err.Message);
  }
  catch (System.Exception err)
  {
    System.Diagnostics.Debug.Fail("Unexpected Error");
    Console.WriteLine("*** Unexpected Error : " + err.Message);
  }
  finally
  {
    myScope.Close();
  }
}

/*
 * Initialize the oscilloscope to a known state.
 * ---------------------------------------------------------------
 */
private static void Initialize()
{
  string strResults;

  // Get and display the device's *IDN? string.
  strResults = myScope.DoQueryString("*IDN?");
  Console.WriteLine("*IDN? result is: {0}", strResults);

  // Clear status and load the default setup.
  myScope.DoCommand("*CLS");
  myScope.DoCommand("*RST");
}

/*
 * Capture the waveform.
 * ---------------------------------------------------------------
 */
private static void Capture()
{
  // Use auto-scale to automatically configure oscilloscope.
  myScope.DoCommand(":AUToscale");

  // Set trigger mode (EDGE, PULSe, PATTern, etc., and input source.
  myScope.DoCommand(":TRIGger:MODE EDGE");
  Console.WriteLine("Trigger mode: {0}",
      myScope.DoQueryString(":TRIGger:MODE?"));

  // Set EDGE trigger parameters.
  myScope.DoCommand(":TRIGger:EDGE:SOURce CHANnel1");
  Console.WriteLine("Trigger edge source: {0}",
      myScope.DoQueryString(":TRIGger:EDGE:SOURce?"));

  myScope.DoCommand(":TRIGger:EDGE:LEVel 1.5");
  Console.WriteLine("Trigger edge level: {0}",
      myScope.DoQueryString(":TRIGger:EDGE:LEVel?"));
```

```
myScope.DoCommand(":TRIGger:EDGE:SLOPe POSitive");
Console.WriteLine("Trigger edge slope: {0}",
    myScope.DoQueryString(":TRIGger:EDGE:SLOPe?"));

// Save oscilloscope configuration.
byte[] ResultsArray;   // Results array.
int nLength;   // Number of bytes returned from instrument.
string strPath;

// Query and read setup string.
ResultsArray = myScope.DoQueryIEEEBlock(":SYSTem:SETup?");
nLength = ResultsArray.Length;

// Write setup string to file.
strPath = "c:\\scope\\config\\setup.scp";
FileStream fStream = File.Open(strPath, FileMode.Create);
fStream.Write(ResultsArray, 0, nLength);
fStream.Close();
Console.WriteLine("Setup bytes saved: {0}", nLength);

// Change settings with individual commands:

// Set vertical scale and offset.
myScope.DoCommand(":CHANnel1:SCALe 0.05");
Console.WriteLine("Channel 1 vertical scale: {0}",
    myScope.DoQueryString(":CHANnel1:SCALe?"));

myScope.DoCommand(":CHANnel1:OFFSet -1.5");
Console.WriteLine("Channel 1 vertical offset: {0}",
    myScope.DoQueryString(":CHANnel1:OFFSet?"));

// Set horizontal scale and offset.
myScope.DoCommand(":TIMebase:SCALe 0.0002");
Console.WriteLine("Timebase scale: {0}",
    myScope.DoQueryString(":TIMebase:SCALe?"));

myScope.DoCommand(":TIMebase:POSition 0.0");
Console.WriteLine("Timebase position: {0}",
    myScope.DoQueryString(":TIMebase:POSition?"));

// Set the acquisition type (NORMal, PEAK, or AVERage).
myScope.DoCommand(":ACQuire:TYPE NORMal");
Console.WriteLine("Acquire type: {0}",
    myScope.DoQueryString(":ACQuire:TYPE?"));

// Or, configure by loading a previously saved setup.
byte[] DataArray;
int nBytesWritten;

// Read setup string from file.
strPath = "c:\\scope\\config\\setup.scp";
DataArray = File.ReadAllBytes(strPath);
nBytesWritten = DataArray.Length;

// Restore setup string.
myScope.DoCommandIEEEBlock(":SYSTem:SETup", DataArray);
Console.WriteLine("Setup bytes restored: {0}", nBytesWritten);
```

```
  // Capture an acquisition using :DIGitize.
  myScope.DoCommand(":DIGitize CHANnel1");
}

/*
 * Analyze the captured waveform.
 * ------------------------------------------------------------
 */
private static void Analyze()
{
  byte[] ResultsArray;   // Results array.
  int nLength;   // Number of bytes returned from instrument.
  string strPath;

  // Make a couple of measurements.
  // ------------------------------------------------------------
  myScope.DoCommand(":MEASure:SOURce CHANnel1");
  Console.WriteLine("Measure source: {0}",
      myScope.DoQueryString(":MEASure:SOURce?"));

  double fResult;
  myScope.DoCommand(":MEASure:FREQuency");
  fResult = myScope.DoQueryNumber(":MEASure:FREQuency?");
  Console.WriteLine("Frequency: {0:F4} kHz", fResult / 1000);

  myScope.DoCommand(":MEASure:VAMPlitude");
  fResult = myScope.DoQueryNumber(":MEASure:VAMPlitude?");
  Console.WriteLine("Vertical amplitude: {0:F2} V", fResult);

  // Download the screen image.
  // ------------------------------------------------------------

  // Get the screen data.
  ResultsArray =
      myScope.DoQueryIEEEBlock(":DISPlay:DATA? PNG");
  nLength = ResultsArray.Length;

  // Store the screen data to a file.
  strPath = "c:\\scope\\data\\screen.png";
  FileStream fStream = File.Open(strPath, FileMode.Create);
  fStream.Write(ResultsArray, 0, nLength);
  fStream.Close();
  Console.WriteLine("Screen image ({0} bytes) written to {1}",
      nLength, strPath);

  // Download waveform data.
  // ------------------------------------------------------------

  // Set the waveform points mode.
  myScope.DoCommand(":WAVeform:POINts:MODE RAW");
  Console.WriteLine("Waveform points mode: {0}",
      myScope.DoQueryString(":WAVeform:POINts:MODE?"));

  // Get the number of waveform points available.
  Console.WriteLine("Waveform points available: {0}",
      myScope.DoQueryString(":WAVeform:POINts?"));
```

```
// Set the waveform source.
myScope.DoCommand(":WAVeform:SOURce CHANnel1");
Console.WriteLine("Waveform source: {0}",
    myScope.DoQueryString(":WAVeform:SOURce?"));

// Choose the format of the data returned (WORD, BYTE, ASCII):
myScope.DoCommand(":WAVeform:FORMat WORD");
Console.WriteLine("Waveform format: {0}",
    myScope.DoQueryString(":WAVeform:FORMat?"));

// Specify the byte order in the word format data:
myScope.DoCommand(":WAVeform:BYTeorder LSBF");
Console.WriteLine("Waveform byte order: {0}",
    myScope.DoQueryString(":WAVeform:BYTeorder?"));

// Display the waveform settings:
double[] fResultsArray;
fResultsArray = myScope.DoQueryNumbers(":WAVeform:PREamble?");

double fFormat = fResultsArray[0];
if (fFormat == 0.0)
{
  Console.WriteLine("Waveform format: BYTE");
}
else if (fFormat == 1.0)
{
  Console.WriteLine("Waveform format: WORD");
}
else if (fFormat == 2.0)
{
  Console.WriteLine("Waveform format: ASCii");
}

double fType = fResultsArray[1];
if (fType == 0.0)
{
  Console.WriteLine("Acquire type: NORMal");
}
else if (fType == 1.0)
{
  Console.WriteLine("Acquire type: PEAK");
}
else if (fType == 2.0)
{
  Console.WriteLine("Acquire type: AVERage");
}

double fPoints = fResultsArray[2];
Console.WriteLine("Waveform points: {0:e}", fPoints);

double fCount = fResultsArray[3];
Console.WriteLine("Waveform average count: {0:e}", fCount);

double fXincrement = fResultsArray[4];
Console.WriteLine("Waveform X increment: {0:e}", fXincrement);
```

```
        double fXorigin = fResultsArray[5];
        Console.WriteLine("Waveform X origin: {0:e}", fXorigin);

        double fXreference = fResultsArray[6];
        Console.WriteLine("Waveform X reference: {0:e}", fXreference);

        double fYincrement = fResultsArray[7];
        Console.WriteLine("Waveform Y increment: {0:e}", fYincrement);

        double fYorigin = fResultsArray[8];
        Console.WriteLine("Waveform Y origin: {0:e}", fYorigin);

        double fYreference = fResultsArray[9];
        Console.WriteLine("Waveform Y reference: {0:e}", fYreference);

        // Read waveform data.
        ResultsArray = myScope.DoQueryIEEEBlock(":WAVeform:DATA?");
        nLength = ResultsArray.Length;
        Console.WriteLine("Number of data values: {0}", nLength / 2);

        // Set up output file:
        strPath = "c:\\scope\\data\\waveform_data.csv";
        if (File.Exists(strPath)) File.Delete(strPath);

        // Open file for output.
        StreamWriter writer = File.CreateText(strPath);

        // Output waveform data in CSV format.
        for (int i = 0; i < nLength - 1; i+=2)
          writer.WriteLine("{0:f9}, {1:f6}",
              fXorigin + ((float)(i / 2) * fXincrement),
              (((float)BitConverter.ToUInt16(ResultsArray, i) - fYreferenc
e)
              * fYincrement) + fYorigin);

        // Close output file.
        writer.Close();
        Console.WriteLine("Waveform format WORD data written to {0}",
            strPath);
      }
    }

    class VisaComInstrument
    {
      private ResourceManagerClass m_ResourceManager;
      private FormattedIO488Class m_IoObject;
      private string m_strVisaAddress;

      // Constructor.
      public VisaComInstrument(string strVisaAddress)
      {
        // Save VISA address in member variable.
        m_strVisaAddress = strVisaAddress;

        // Open the default VISA COM IO object.
        OpenIo();
```

```
  // Clear the interface.
  m_IoObject.IO.Clear();
}

public void DoCommand(string strCommand)
{
  // Send the command.
  m_IoObject.WriteString(strCommand, true);

  // Check for inst errors.
  CheckInstrumentErrors(strCommand);
}

public void DoCommandIEEEBlock(string strCommand,
    byte[] DataArray)
{
  // Send the command to the device.
  m_IoObject.WriteIEEEBlock(strCommand, DataArray, true);

  // Check for inst errors.
  CheckInstrumentErrors(strCommand);
}

public string DoQueryString(string strQuery)
{
  // Send the query.
  m_IoObject.WriteString(strQuery, true);

  // Get the result string.
  string strResults;
  strResults = m_IoObject.ReadString();

  // Check for inst errors.
  CheckInstrumentErrors(strQuery);

  // Return results string.
  return strResults;
}

public double DoQueryNumber(string strQuery)
{
  // Send the query.
  m_IoObject.WriteString(strQuery, true);

  // Get the result number.
  double fResult;
  fResult = (double)m_IoObject.ReadNumber(
    IEEEASCIIType.ASCIIType_R8, true);

  // Check for inst errors.
  CheckInstrumentErrors(strQuery);

  // Return result number.
  return fResult;
}

public double[] DoQueryNumbers(string strQuery)
```

```
{
  // Send the query.
  m_IoObject.WriteString(strQuery, true);

  // Get the result numbers.
  double[] fResultsArray;
  fResultsArray = (double[])m_IoObject.ReadList(
    IEEEASCIIType.ASCIIType_R8, ",;");

  // Check for inst errors.
  CheckInstrumentErrors(strQuery);

  // Return result numbers.
  return fResultsArray;
}

public byte[] DoQueryIEEEBlock(string strQuery)
{
  // Send the query.
  m_IoObject.WriteString(strQuery, true);

  // Get the results array.
  System.Threading.Thread.Sleep(2000); // Delay before reading.
  byte[] ResultsArray;
  ResultsArray = (byte[])m_IoObject.ReadIEEEBlock(
    IEEEBinaryType.BinaryType_UI1, false, true);

  // Check for inst errors.
  CheckInstrumentErrors(strQuery);

  // Return results array.
  return ResultsArray;
}

private void CheckInstrumentErrors(string strCommand)
{
  // Check for instrument errors.
  string strInstrumentError;
  bool bFirstError = true;

  do   // While not "0,No error".
  {
    m_IoObject.WriteString(":SYSTem:ERRor?", true);
    strInstrumentError = m_IoObject.ReadString();

    if (!strInstrumentError.ToString().StartsWith("+0,"))
    {
      if (bFirstError)
      {
        Console.WriteLine("ERROR(s) for command '{0}': ",
          strCommand);
        bFirstError = false;
      }
      Console.Write(strInstrumentError);
    }
  } while (!strInstrumentError.ToString().StartsWith("+0,"));
}
```

```
    private void OpenIo()
    {
      m_ResourceManager = new ResourceManagerClass();
      m_IoObject = new FormattedIO488Class();

      // Open the default VISA COM IO object.
      try
      {
        m_IoObject.IO =
          (IMessage)m_ResourceManager.Open(m_strVisaAddress,
          AccessMode.NO_LOCK, 0, "");
      }
      catch (Exception e)
      {
        Console.WriteLine("An error occurred: {0}", e.Message);
      }
    }

    public void SetTimeoutSeconds(int nSeconds)
    {
      m_IoObject.IO.Timeout = nSeconds * 1000;
    }

    public void Close()
    {
      try
      {
        m_IoObject.IO.Close();
      }
      catch { }

      try
      {
        Marshal.ReleaseComObject(m_IoObject);
      }
      catch { }

      try
      {
        Marshal.ReleaseComObject(m_ResourceManager);
      }
      catch { }
    }
  }
}
```

VISA COM Example in Visual Basic .NET

To compile and run this example in Microsoft Visual Studio 2019:

1. Open Visual Studio.
2. Create a new Visual Basic, Windows, Console Application project using the **.NET Framework 4.7.2**.
3. Cut-and-paste the code that follows into the Visual Basic .NET source file.

4 Edit the program to use the VISA address of your oscilloscope.

5 Add a reference to the VISA COM 5.14 Type Library:

   a Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment.

   b Choose **Add > Reference...**.

   c In the Reference Manager dialog box, select the **COM** hierarchy on the left.

   d Select **VISA COM 5.14 Type Library**; then click **OK**.

   e Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment and choose **Properties**; then, select "InfiniiVision.VisaComInstrumentApp" as the **Startup object**.

6 In the Solution Explorer, expand References, and select **VisaComLib**. Then, in the Properties window, change **Embed Interop Types** to **False**.

7 Build and run the program.

For more information, see the VISA COM Help that comes with Keysight IO Libraries Suite.

```
'
' Keysight VISA COM Example in Visual Basic .NET
' ----------------------------------------------------------------------
' This program illustrates a few commonly used programming
' features of your Keysight oscilloscope.
' ----------------------------------------------------------------------

Imports System
Imports System.IO
Imports System.Text
Imports Ivi.Visa.Interop
Imports System.Runtime.InteropServices

Namespace InfiniiVision
  Class VisaComInstrumentApp
    Private Shared myScope As VisaComInstrument

    Public Shared Sub Main(ByVal args As String())
      Try
        myScope = New _
          VisaComInstrument("USB0::0x2A8D::0x4704::Unset::0::INSTR")
        myScope.SetTimeoutSeconds(10)

        ' Initialize - start from a known state.
        Initialize()

        ' Capture data.
        Capture()

        ' Analyze the captured waveform.
        Analyze()

      Catch err As System.ApplicationException
```

```
    Console.WriteLine("*** VISA Error Message : " + err.Message)
  Catch err As System.SystemException
    Console.WriteLine("*** System Error Message : " + err.Message)
  Catch err As System.Exception
    System.Diagnostics.Debug.Fail("Unexpected Error")
    Console.WriteLine("*** Unexpected Error : " + err.Message)
  Finally
    myScope.Close()
  End Try
End Sub

' Initialize the oscilloscope to a known state.
' ---------------------------------------------------------------

Private Shared Sub Initialize()
  Dim strResults As String

  ' Get and display the device's *IDN? string.
  strResults = myScope.DoQueryString("*IDN?")
  Console.WriteLine("*IDN? result is: {0}", strResults)

  ' Clear status and load the default setup.
  myScope.DoCommand("*CLS")
  myScope.DoCommand("*RST")

End Sub

' Capture the waveform.
' ---------------------------------------------------------------

Private Shared Sub Capture()

  ' Use auto-scale to automatically configure oscilloscope.
  myScope.DoCommand(":AUToscale")

  ' Set trigger mode (EDGE, PULSe, PATTern, etc., and input source.
  myScope.DoCommand(":TRIGger:MODE EDGE")
  Console.WriteLine("Trigger mode: {0}",
      myScope.DoQueryString(":TRIGger:MODE?"))

  ' Set EDGE trigger parameters.
  myScope.DoCommand(":TRIGger:EDGE:SOURce CHANnel1")
  Console.WriteLine("Trigger edge source: {0}",
      myScope.DoQueryString(":TRIGger:EDGE:SOURce?"))

  myScope.DoCommand(":TRIGger:EDGE:LEVel 1.5")
  Console.WriteLine("Trigger edge level: {0}",
      myScope.DoQueryString(":TRIGger:EDGE:LEVel?"))

  myScope.DoCommand(":TRIGger:EDGE:SLOPe POSitive")
  Console.WriteLine("Trigger edge slope: {0}",
      myScope.DoQueryString(":TRIGger:EDGE:SLOPe?"))

  ' Save oscilloscope configuration.
  Dim ResultsArray As Byte()   ' Results array.
  Dim nLength As Integer   ' Number of bytes returned from inst.
  Dim strPath As String
```

```
    Dim fStream As FileStream

    ' Query and read setup string.
    ResultsArray = myScope.DoQueryIEEEBlock(":SYSTem:SETup?")
    nLength = ResultsArray.Length

    ' Write setup string to file.
    strPath = "c:\scope\config\setup.scp"
    fStream = File.Open(strPath, FileMode.Create)
    fStream.Write(ResultsArray, 0, nLength)
    fStream.Close()
    Console.WriteLine("Setup bytes saved: {0}", nLength)

    ' Change settings with individual commands:

    ' Set vertical scale and offset.
    myScope.DoCommand(":CHANnel1:SCALe 0.05")
    Console.WriteLine("Channel 1 vertical scale: {0}",
        myScope.DoQueryString(":CHANnel1:SCALe?"))

    myScope.DoCommand(":CHANnel1:OFFSet -1.5")
    Console.WriteLine("Channel 1 vertical offset: {0}",
        myScope.DoQueryString(":CHANnel1:OFFSet?"))

    ' Set horizontal scale and offset.
    myScope.DoCommand(":TIMebase:SCALe 0.0002")
    Console.WriteLine("Timebase scale: {0}",
        myScope.DoQueryString(":TIMebase:SCALe?"))

    myScope.DoCommand(":TIMebase:POSition 0.0")
    Console.WriteLine("Timebase position: {0}",
        myScope.DoQueryString(":TIMebase:POSition?"))

    ' Set the acquisition type (NORMal, PEAK, or AVERage).
    myScope.DoCommand(":ACQuire:TYPE NORMal")
    Console.WriteLine("Acquire type: {0}",
        myScope.DoQueryString(":ACQuire:TYPE?"))

    ' Or, configure by loading a previously saved setup.
    Dim DataArray As Byte()
    Dim nBytesWritten As Integer

    ' Read setup string from file.
    strPath = "c:\scope\config\setup.scp"
    DataArray = File.ReadAllBytes(strPath)
    nBytesWritten = DataArray.Length

    ' Restore setup string.
    myScope.DoCommandIEEEBlock(":SYSTem:SETup", DataArray)
    Console.WriteLine("Setup bytes restored: {0}", nBytesWritten)

    ' Capture an acquisition using :DIGitize.
    myScope.DoCommand(":DIGitize CHANnel1")

End Sub

' Analyze the captured waveform.
```

```
' ----------------------------------------------------------------

Private Shared Sub Analyze()

  Dim fResult As Double
  Dim ResultsArray As Byte()   ' Results array.
  Dim nLength As Integer   ' Number of bytes returned from inst.
  Dim strPath As String

  ' Make a couple of measurements.
  ' ----------------------------------------------------------------
  myScope.DoCommand(":MEASure:SOURce CHANnel1")
  Console.WriteLine("Measure source: {0}",
      myScope.DoQueryString(":MEASure:SOURce?"))

  myScope.DoCommand(":MEASure:FREQuency")
  fResult = myScope.DoQueryNumber(":MEASure:FREQuency?")
  Console.WriteLine("Frequency: {0:F4} kHz", fResult / 1000)

  myScope.DoCommand(":MEASure:VAMPlitude")
  fResult = myScope.DoQueryNumber(":MEASure:VAMPlitude?")
  Console.WriteLine("Vertical amplitude: {0:F2} V", fResult)

  ' Download the screen image.
  ' ----------------------------------------------------------------

  ' Get the screen data.
  ResultsArray = myScope.DoQueryIEEEBlock(":DISPlay:DATA? PNG")
  nLength = ResultsArray.Length

  ' Store the screen data to a file.
  strPath = "c:\scope\data\screen.png"
  Dim fStream As FileStream
  fStream = File.Open(strPath, FileMode.Create)
  fStream.Write(ResultsArray, 0, nLength)
  fStream.Close()
  Console.WriteLine("Screen image ({0} bytes) written to {1}",
      nLength, strPath)

  ' Download waveform data.
  ' ----------------------------------------------------------------

  ' Set the waveform points mode.
  myScope.DoCommand(":WAVeform:POINts:MODE RAW")
  Console.WriteLine("Waveform points mode: {0}",
      myScope.DoQueryString(":WAVeform:POINts:MODE?"))

  ' Get the number of waveform points available.
  myScope.DoCommand(":WAVeform:POINts 10240")
  Console.WriteLine("Waveform points available: {0}",
      myScope.DoQueryString(":WAVeform:POINts?"))

  ' Set the waveform source.
  myScope.DoCommand(":WAVeform:SOURce CHANnel1")
  Console.WriteLine("Waveform source: {0}",
      myScope.DoQueryString(":WAVeform:SOURce?"))
```

```
' Choose the format of the data returned (WORD, BYTE, ASCII):
myScope.DoCommand(":WAVeform:FORMat WORD")
Console.WriteLine("Waveform format: {0}",
    myScope.DoQueryString(":WAVeform:FORMat?"))

' Specify the byte order in the word format data:
myScope.DoCommand(":WAVeform:BYTeorder LSBF")
Console.WriteLine("Waveform byte order: {0}",
    myScope.DoQueryString(":WAVeform:BYTeorder?"))

' Display the waveform settings:
Dim fResultsArray As Double()
fResultsArray = myScope.DoQueryNumbers(":WAVeform:PREamble?")

Dim fFormat As Double = fResultsArray(0)
If fFormat = 0 Then
  Console.WriteLine("Waveform format: BYTE")
ElseIf fFormat = 1 Then
  Console.WriteLine("Waveform format: WORD")
ElseIf fFormat = 2 Then
  Console.WriteLine("Waveform format: ASCii")
End If

Dim fType As Double = fResultsArray(1)
If fType = 0 Then
  Console.WriteLine("Acquire type: NORMal")
ElseIf fType = 1 Then
  Console.WriteLine("Acquire type: PEAK")
ElseIf fType = 2 Then
  Console.WriteLine("Acquire type: AVERage")
End If

Dim fPoints As Double = fResultsArray(2)
Console.WriteLine("Waveform points: {0:e}", fPoints)

Dim fCount As Double = fResultsArray(3)
Console.WriteLine("Waveform average count: {0:e}", fCount)

Dim fXincrement As Double = fResultsArray(4)
Console.WriteLine("Waveform X increment: {0:e}", fXincrement)

Dim fXorigin As Double = fResultsArray(5)
Console.WriteLine("Waveform X origin: {0:e}", fXorigin)

Dim fXreference As Double = fResultsArray(6)
Console.WriteLine("Waveform X reference: {0:e}", fXreference)

Dim fYincrement As Double = fResultsArray(7)
Console.WriteLine("Waveform Y increment: {0:e}", fYincrement)

Dim fYorigin As Double = fResultsArray(8)
Console.WriteLine("Waveform Y origin: {0:e}", fYorigin)

Dim fYreference As Double = fResultsArray(9)
Console.WriteLine("Waveform Y reference: {0:e}", fYreference)

' Get the waveform data.
```

```
        ResultsArray = myScope.DoQueryIEEEBlock(":WAVeform:DATA?")
        nLength = ResultsArray.Length
        Console.WriteLine("Number of data values: {0}", nLength / 2)

        ' Set up output file:
        strPath = "c:\scope\data\waveform_data.csv"
        If File.Exists(strPath) Then
          File.Delete(strPath)
        End If

        ' Open file for output.
        Dim writer As StreamWriter = File.CreateText(strPath)

        ' Output waveform data in CSV format.
        For index As Integer = 0 To nLength - 1 Step 2
          ' Write time value, voltage value.
          writer.WriteLine("{0:f9}, {1:f6}",
              fXorigin + (CSng(index / 2) * fXincrement),
              ((CSng(BitConverter.ToUInt16(ResultsArray, index)) - fYrefer
ence) _
              * fYincrement) + fYorigin)
        Next

        ' Close output file.
        writer.Close()
        Console.WriteLine("Waveform format WORD data written to {0}",
            strPath)

      End Sub

    End Class

    Class VisaComInstrument
      Private m_ResourceManager As ResourceManagerClass
      Private m_IoObject As FormattedIO488Class
      Private m_strVisaAddress As String

      ' Constructor.
      Public Sub New(ByVal strVisaAddress As String)

        ' Save VISA address in member variable.
        m_strVisaAddress = strVisaAddress

        ' Open the default VISA COM IO object.
        OpenIo()

        ' Clear the interface.
        m_IoObject.IO.Clear()

      End Sub

      Public Sub DoCommand(ByVal strCommand As String)

        ' Send the command.
        m_IoObject.WriteString(strCommand, True)

        ' Check for inst errors.
```

```
  CheckInstrumentErrors(strCommand)

End Sub

Public Sub DoCommandIEEEBlock(ByVal strCommand As String,
    ByVal DataArray As Byte())

  ' Send the command to the device.
  m_IoObject.WriteIEEEBlock(strCommand, DataArray, True)

  ' Check for inst errors.
  CheckInstrumentErrors(strCommand)

End Sub

Public Function DoQueryString(ByVal strQuery As String) As String
  ' Send the query.
  m_IoObject.WriteString(strQuery, True)

  ' Get the result string.
  Dim strResults As String
  strResults = m_IoObject.ReadString()

  ' Check for inst errors.
  CheckInstrumentErrors(strQuery)

  ' Return results string.
  Return strResults
End Function

Public Function DoQueryNumber(ByVal strQuery As String) As Double
  ' Send the query.
  m_IoObject.WriteString(strQuery, True)

  ' Get the result number.
  Dim fResult As Double
  fResult =
      CDbl(m_IoObject.ReadNumber(IEEEASCIIType.ASCIIType_R8, True))

  ' Check for inst errors.
  CheckInstrumentErrors(strQuery)

  ' Return result number.
  Return fResult
End Function

Public Function DoQueryNumbers(ByVal strQuery As String) As _
    Double()
  ' Send the query.
  m_IoObject.WriteString(strQuery, True)

  ' Get the result numbers.
  Dim fResultsArray As Double()
  fResultsArray =
      m_IoObject.ReadList(IEEEASCIIType.ASCIIType_R8, ",;")

  ' Check for inst errors.
```

```
  CheckInstrumentErrors(strQuery)

  ' Return result numbers.
  Return fResultsArray
End Function

Public _
    Function DoQueryIEEEBlock(ByVal strQuery As String) As Byte()
  ' Send the query.
  m_IoObject.WriteString(strQuery, True)

  ' Get the results array.
  System.Threading.Thread.Sleep(2000) ' Delay before reading data.
  Dim ResultsArray As Byte()
  ResultsArray =
      m_IoObject.ReadIEEEBlock(IEEEBinaryType.BinaryType_UI1,
      False, True)

  ' Check for inst errors.
  CheckInstrumentErrors(strQuery)

  ' Return results array.
  Return ResultsArray
End Function

Private Sub CheckInstrumentErrors(ByVal strCommand As String)
  ' Check for instrument errors.
  Dim strInstrumentError As String
  Dim bFirstError As Boolean = True
  Do   ' While not "0,No error".
    m_IoObject.WriteString(":SYSTem:ERRor?", True)
    strInstrumentError = m_IoObject.ReadString()

    If Not strInstrumentError.ToString().StartsWith("+0,") Then
      If bFirstError Then
        Console.WriteLine("ERROR(s) for command '{0}': ",
            strCommand)
        bFirstError = False
      End If
      Console.Write(strInstrumentError)
    End If
  Loop While Not strInstrumentError.ToString().StartsWith("+0,")
End Sub

Private Sub OpenIo()
  m_ResourceManager = New ResourceManagerClass()
  m_IoObject = New FormattedIO488Class()

  ' Open the default VISA COM IO object.
  Try
    m_IoObject.IO =
        DirectCast(m_ResourceManager.Open(m_strVisaAddress,
            AccessMode.NO_LOCK, 0, ""), IMessage)
  Catch e As Exception
    Console.WriteLine("An error occurred: {0}", e.Message)
  End Try
End Sub
```

```
    Public Sub SetTimeoutSeconds(ByVal nSeconds As Integer)
      m_IoObject.IO.Timeout = nSeconds * 1000
    End Sub

    Public Sub Close()
      Try
        m_IoObject.IO.Close()
      Catch
      End Try

      Try
        Marshal.ReleaseComObject(m_IoObject)
      Catch
      End Try

      Try
        Marshal.ReleaseComObject(m_ResourceManager)
      Catch
      End Try
    End Sub
  End Class
End Namespace
```

VISA COM Example in Python

You can use the Python programming language with the "comtypes" package to control Keysight oscilloscopes.

The Python language and "comtypes" package can be downloaded from the web at http://www.python.org/ and https://pypi.org/project/comtypes/, respectively.

To run this example with Python and "comtypes":

1. Cut-and-paste the code that follows into a file named "example.py".
2. Edit the program to use the VISA address of your oscilloscope.
3. If "python.exe" can be found via your PATH environment variable, open a Command Prompt window; then, change to the folder that contains the "example.py" file, and enter:

```
python example.py
```

```
#!python3
# -*- coding: utf-8 -*-

# Keysight VISA COM Example in Python using "comtypes"
# ******************************************************
# This program illustrates a few commonly-used programming
# features of your Keysight oscilloscope.
# ******************************************************

# Import Python modules.
# ------------------------------------------------------------
import sys
import array
```

```
from comtypes.client import GetModule
from comtypes.client import CreateObject

# Run GetModule once to generate comtypes.gen.VisaComLib.
if not hasattr(sys, "frozen"):
 GetModule("C:\Program Files (x86)\IVI Foundation\VISA\VisaCom\
GlobMgr.dll")

import comtypes.gen.VisaComLib as VisaComLib


# Global variables.
# --------------------------------------------------------
visa_address = "USB0::0x2A8D::0x4704::Unset::0::INSTR"
input_channel = "CHANnel1"
# input_channel = "CHANnel2"
setup_file_name = "setup.scp"
screen_image_file_name = "screen_image.png"
waveform_data_file_name = "waveform_data.csv"


# ========================================================
# Initialize:
# ========================================================
def initialize():

 # Get and display the device's *IDN? string.
 idn_string = do_query_string("*IDN?")
 print(f"Identification string: '{idn_string}'")

 # Clear status and load the default setup.
 do_command("*CLS")
 do_command("*RST")


# ========================================================
# Capture:
# ========================================================
def capture():

 # Use auto-scale to automatically set up oscilloscope.
 print("Autoscale.")
 do_command(":AUToscale")

 # Set trigger mode.
 do_command(":TRIGger:MODE EDGE")
 qresult = do_query_string(":TRIGger:MODE?")
 print(f"Trigger mode: {qresult}")

 # Set EDGE trigger parameters.
 do_command(f":TRIGger:EDGE:SOURce {input_channel}")
 qresult = do_query_string(":TRIGger:EDGE:SOURce?")
 print(f"Trigger edge source: {qresult}")

 do_command(":TRIGger:EDGE:LEVel 1.5")
 qresult = do_query_string(":TRIGger:EDGE:LEVel?")
```

```
print(f"Trigger edge level: {qresult}")

do_command(":TRIGger:EDGE:SLOPe POSitive")
qresult = do_query_string(":TRIGger:EDGE:SLOPe?")
print(f"Trigger edge slope: {qresult}")

# Save oscilloscope setup.
setup_bytes = do_query_ieee_block(":SYSTem:SETup?")
setup_bytes_length = len(setup_bytes)
f = open(setup_file_name, "wb")
f.write(bytearray(setup_bytes))
f.close()
print(f"Setup bytes saved: {setup_bytes_length}")

# Change oscilloscope settings with individual commands:

# Set vertical scale and offset.
do_command(f":{input_channel}:SCALe 0.05")
qresult = do_query_string(f":{input_channel}:SCALe?")
print(f"{input_channel} vertical scale: {qresult}")

do_command(f":{input_channel}:OFFSet -1.5")
qresult = do_query_string(f":{input_channel}:OFFSet?")
print(f"{input_channel} offset: {qresult}")

# Set horizontal scale and offset.
do_command(":TIMebase:SCALe 0.0002")
qresult = do_query_string(":TIMebase:SCALe?")
print(f"Timebase scale: {qresult}")

do_command(":TIMebase:POSition 0.0")
qresult = do_query_string(":TIMebase:POSition?")
print(f"Timebase position: {qresult}")

# Set the acquisition type.
do_command(":ACQuire:TYPE NORMal")
qresult = do_query_string(":ACQuire:TYPE?")
print(f"Acquire type: {qresult}")

# Or, set up oscilloscope by loading a previously saved setup.
f = open(setup_file_name, "rb")
setup_bytes = f.read()
f.close()
do_command_ieee_block(":SYSTem:SETup", array.array('B', setup_bytes))
print(f"Setup bytes restored: {len(setup_bytes)}")

# Capture an acquisition using :DIGitize.
do_command(f":DIGitize {input_channel}")


# ============================================================
# Analyze:
# ============================================================
def analyze():

 # Make measurements.
 # ------------------------------------------------------------
```

```
do_command(f":MEASure:SOURce {input_channel}")
qresult = do_query_string(":MEASure:SOURce?")
print(f"Measure source: {qresult}")

do_command(":MEASure:FREQuency")
qresult = do_query_string(":MEASure:FREQuency?")
print(f"Measured frequency on {input_channel}: {qresult}")

do_command(":MEASure:VAMPlitude")
qresult = do_query_string(":MEASure:VAMPlitude?")
print(f"Measured vertical amplitude on {input_channel}: {qresult}")

# Download the screen image.
# -------------------------------------------------------

screen_bytes = do_query_ieee_block(":DISPlay:DATA? PNG")

# Save display data values to file.
f = open(screen_image_file_name, "wb")
f.write(bytearray(screen_bytes))
f.close()
print(f"Screen image written to {screen_image_file_name}.")

# Download waveform data.
# -------------------------------------------------------

# Set the waveform source.
do_command(f":WAVeform:SOURce {input_channel}")
qresult = do_query_string(":WAVeform:SOURce?")
print(f"Waveform source: {qresult}")

# Set the waveform points mode.
do_command(":WAVeform:POINts:MODE RAW")
qresult = do_query_string(":WAVeform:POINts:MODE?")
print(f"Waveform points mode: {qresult}")

# Get the number of waveform points available.
do_command(":WAVeform:POINts 10240")
qresult = do_query_string(":WAVeform:POINts?")
print(f"Waveform points available: {qresult}")

# Choose the format of the data returned (BYTE or WORD):
do_command(f":WAVeform:FORMat WORD")
qresult = do_query_string(":WAVeform:FORMat?")
print(f"Waveform format: {qresult}")

# Display the waveform settings from preamble:
wav_form_dict = {
 0 : "BYTE",
 1 : "WORD",
 4 : "ASCii",
}
acq_type_dict = {
 0 : "NORMal",
 1 : "PEAK",
 2 : "AVERage",
}
```

```
 (
  wav_form,
  acq_type,
  wfmpts,
  avgcnt,
  x_increment,
  x_origin,
  x_reference,
  y_increment,
  y_origin,
  y_reference
 ) = do_query_numbers(":WAVeform:PREamble?")

 print(f"Waveform format: {wav_form_dict[int(wav_form)]}")
 print(f"Acquire type: {acq_type_dict[int(acq_type)]}")
 print(f"Waveform points desired: {wfmpts}")
 print(f"Waveform average count: {avgcnt}")
 print(f"Waveform X reference: {x_reference}")   # Always 0.

 # Get numeric values for later calculations.
 x_increment = do_query_number(":WAVeform:XINCrement?")
 print(f"Waveform X increment: {x_increment}")
 x_origin = do_query_number(":WAVeform:XORigin?")
 print(f"Waveform X origin: {x_origin}")
 y_increment = do_query_number(":WAVeform:YINCrement?")
 print(f"Waveform Y increment: {y_increment}")
 y_origin = do_query_number(":WAVeform:YORigin?")
 print(f"Waveform Y origin: {y_origin}")
 y_reference = do_query_number(":WAVeform:YREFerence?")
 print(f"Waveform Y reference: {y_reference}")

 # Specify the byte order in WORD data.
 do_command(":WAVeform:BYTeorder LSBF")
 qresult = do_query_string(":WAVeform:BYTeorder?")
 print(f"Waveform byte order for WORD data: {qresult}")

 # Get the waveform data bytes.
 data_bytes = do_query_ieee_block(":WAVeform:DATA?")
 data_bytes_length = len(data_bytes)
 print(f"Byte count: {data_bytes_length}")

 # Get the waveform data values.
 data_bytes_array = array.array('B', data_bytes)   # Array of unsigned b
ytes.
 values = array.array('H')   # Interpret as array of unsigned shorts.
 values.frombytes(data_bytes_array)
 print(f"Number of data values: {len(values)}")

 # Save waveform data values to CSV file.
 f = open(waveform_data_file_name, "w")

 for i in range(0, len(values) - 1):
  time_val = x_origin + (i * x_increment)
  voltage = ((values[i] - y_reference) * y_increment) + y_origin
  f.write(f"{time_val:E}, {voltage:f}\n")
```

```
 # Close output file.
 f.close()
 print(f"Waveform format WORD data written to {waveform_data_file_name}.
")


# ========================================================
# Send a command and check for errors:
# ========================================================
def do_command(command):
 myScope.WriteString(command, True)
 check_instrument_errors(command)


# ========================================================
# Send a command and check for errors:
# ========================================================
def do_command_ieee_block(command, data):
 myScope.WriteIEEEBlock(command, data, True)
 check_instrument_errors(command)


# ========================================================
# Send a query, check for errors, return string:
# ========================================================
def do_query_string(query):
 myScope.WriteString(query, True)
 result = myScope.ReadString()
 check_instrument_errors(query)
 return result.strip()


# ========================================================
# Send a query, check for errors, return string:
# ========================================================
def do_query_ieee_block(query):
 myScope.WriteString(query, True)
 result = myScope.ReadIEEEBlock(VisaComLib.BinaryType_UI1, \
   False, True)
 check_instrument_errors(query)
 return result


# ========================================================
# Send a query, check for errors, return values:
# ========================================================
def do_query_number(query):
 myScope.WriteString(query, True)
 result = myScope.ReadNumber(VisaComLib.ASCIIType_R8, True)
 check_instrument_errors(query)
 return result


# ========================================================
# Send a query, check for errors, return values:
# ========================================================
def do_query_numbers(query):
```

```
 myScope.WriteString(query, True)
 result = myScope.ReadList(VisaComLib.ASCIIType_R8, ",;")
 check_instrument_errors(query)
 return result


# ============================================================
# Check for instrument errors:
# ============================================================
def check_instrument_errors(command):

 while True:
  myScope.WriteString(":SYSTem:ERRor?", True)
  error_string = myScope.ReadString()
  if error_string:   # If there is an error string value.

   if error_string.find("+0,", 0, 3) == -1:   # Not "No error".
    print(f"ERROR: {error_string}, command: '{command}'")
    print("Exited because of error.")
    sys.exit(1)

   else:   # "No error"
    break

  else:   # :SYSTem:ERRor? should always return string.
   print("ERROR: :SYSTem:ERRor? returned nothing, command: '%s'" \
     % command)
   print("Exited because of error.")
   sys.exit(1)


# ============================================================
# Main program:
# ============================================================
rm = CreateObject("VISA.GlobalRM", \
 interface=VisaComLib.IResourceManager)
myScope = CreateObject("VISA.BasicFormattedIO", \
 interface=VisaComLib.IFormattedIO488)
myScope.IO = \
 rm.Open(visa_address)

# Clear the interface.
myScope.IO.Clear()
print("Interface cleared.")

# Set the Timeout to 15 seconds.
myScope.IO.Timeout = 15000   # 15 seconds.
print("Timeout set to 15000 milliseconds.")

# Initialize the oscilloscope, capture data, and analyze.
initialize()
capture()
analyze()

myScope.IO.Close()
print("End of program.")
sys.exit()
```

VISA Examples

VISA Example in C

To compile and run this example in Microsoft Visual Studio 2019:

1. Open Visual Studio.
2. Create a new Visual C++, Windows, Console, Empty project.
3. Cut-and-paste the code that follows into a file named "example.c" in the project directory.
4. In Visual Studio 2019, right-click the Source Files folder, choose **Add > Add Existing Item...**, select the example.c file, and click **Add**.
5. Edit the program to use the VISA address of your oscilloscope.
6. In the Solution Explorer, right-click the project (not the solution), and choose **Properties**. In the Property Pages dialog box, update these project settings:
   a. Under Configuration Properties, Linker, Input, add "visa32.lib" to the Additional Dependencies field.
   b. Under Configuration Properties, C/C++, Code Generation, select Multi-threaded DLL for the Runtime Library field.
   c. Under Configuration Properties, C/C++, Preprocessor, add "_CRT_SECURE_NO_WARNINGS" to the Preprocessor Definitions.
   d. Under Configuration Properties, VC++ Directories, show directories for **Include Directories**, and add the include directory (for example, Program Files (x86)\IVI Foundation\VISA\WinNT\Include).
   e. Under Configuration Properties, VC++ Directories, show directories for **Library Directories**, and add the library files directory (for example, Program Files (x86)\IVI Foundation\VISA\WinNT\lib\msc).
   f. Click **OK** to close the Property Pages dialog.
7. Build and run the program.

```
/*
 * Keysight VISA Example in C
 * ----------------------------------------------------------------------
 * This program illustrates a few commonly-used programming
 * features of your Keysight oscilloscope.
 */

#include <stdio.h>            /* For printf(). */
#include <string.h>           /* For strcpy(), strcat(). */
```

```
#include <time.h>              /* For clock(). */
#include <visa.h>              /* Keysight VISA routines. */

#define VISA_ADDRESS "USB0::0x2A8D::0x4704::Unset::0::INSTR"
#define IEEEBLOCK_SPACE 5000000

 /* Function prototypes */
void initialize(void);          /* Initialize to known state. */
void capture(void);             /* Capture the waveform. */
void analyze(void);             /* Analyze the captured waveform. */

void do_command(char* command);        /* Send command. */
int do_command_ieeeblock(char* command); /* Command w/IEEE block. */
void do_query_string(char* query);     /* Query for string. */
void do_query_number(char* query);     /* Query for number. */
void do_query_numbers(char* query);    /* Query for numbers. */
int do_query_ieeeblock(char* query);   /* Query for IEEE block. */
void check_instrument_errors();        /* Check for inst errors. */
void error_handler();                  /* VISA error handler. */

/* Global variables */
ViSession defaultRM, vi;        /* Device session ID. */
ViStatus err;                   /* VISA function return value. */
char str_result[256] = { 0 };   /* Result from do_query_string(). */
double num_result;              /* Result from do_query_number(). */
unsigned char ieeeblock_data[IEEEBLOCK_SPACE];   /* Result from
                                   do_query_ieeeblock(). */
double dbl_results[10];         /* Result from do_query_numbers(). */

/* Main Program
 * ------------------------------------------------------------- */
void main(void)
{
    /* Open the default resource manager session. */
    err = viOpenDefaultRM(&defaultRM);
    if (err != VI_SUCCESS) error_handler();

    /* Open the session using the oscilloscope's VISA address. */
    err = viOpen(defaultRM, VISA_ADDRESS, VI_NULL, VI_NULL, &vi);
    if (err != VI_SUCCESS) error_handler();

    /* Set the I/O timeout to fifteen seconds. */
    err = viSetAttribute(vi, VI_ATTR_TMO_VALUE, 15000);
    if (err != VI_SUCCESS) error_handler();

    /* Initialize - start from a known state. */
    initialize();

    /* Capture data. */
    capture();

    /* Analyze the captured waveform. */
    analyze();

    /* Close the vi session and the resource manager session. */
    viClose(vi);
    viClose(defaultRM);
```

```
}

/* Initialize the oscilloscope to a known state.
 * ------------------------------------------------------------------ */
void initialize(void)
{
    /* Clear the interface. */
    err = viClear(vi);
    if (err != VI_SUCCESS) error_handler();

    /* Get and display the device's *IDN? string. */
    do_query_string("*IDN?");
    printf("Oscilloscope *IDN? string: %s\n", str_result);

    /* Clear status and load the default setup. */
    do_command("*CLS");
    do_command("*RST");
}

/* Capture the waveform.
 * ------------------------------------------------------------------ */
void capture(void)
{
    int num_bytes;
    FILE* fp;

    /* Use auto-scale to automatically configure oscilloscope. */
    do_command(":AUToscale");

    /* Set trigger mode (EDGE, PULSe, PATTern, etc., and input source. *
/
    do_command(":TRIGger:MODE EDGE");
    do_query_string(":TRIGger:MODE?");
    printf("Trigger mode: %s\n", str_result);

    /* Set EDGE trigger parameters. */
    do_command(":TRIGger:EDGE:SOURce CHANnel1");
    do_query_string(":TRIGger:EDGE:SOURce?");
    printf("Trigger edge source: %s\n", str_result);

    do_command(":TRIGger:EDGE:LEVel 1.5");
    do_query_string(":TRIGger:EDGE:LEVel?");
    printf("Trigger edge level: %s\n", str_result);

    do_command(":TRIGger:EDGE:SLOPe POSitive");
    do_query_string(":TRIGger:EDGE:SLOPe?");
    printf("Trigger edge slope: %s\n", str_result);

    /* Save oscilloscope configuration. */

    /* Read system setup. */
    num_bytes = do_query_ieeeblock(":SYSTem:SETup?");
    printf("Read setup string query (%d bytes).\n", num_bytes);

    /* Write setup string to file. */
    fp = fopen("c:\\scope\\config\\setup.scp", "wb");
    num_bytes = fwrite(ieeeblock_data, sizeof(unsigned char), num_bytes,
```

```
        fp);
    fclose(fp);
    printf("Wrote setup string (%d bytes) to ", num_bytes);
    printf("c:\\scope\\config\\setup.stp.\n");

    /* Change settings with individual commands:

    /* Set vertical scale and offset. */
    do_command(":CHANnel1:SCALe 0.05");
    do_query_string(":CHANnel1:SCALe?");
    printf("Channel 1 vertical scale: %s\n", str_result);

    do_command(":CHANnel1:OFFSet -1.5");
    do_query_string(":CHANnel1:OFFSet?");
    printf("Channel 1 offset: %s\n", str_result);

    /* Set horizontal scale and offset. */
    do_command(":TIMebase:SCALe 0.0002");
    do_query_string(":TIMebase:SCALe?");
    printf("Timebase scale: %s\n", str_result);

    do_command(":TIMebase:POSition 0.0");
    do_query_string(":TIMebase:POSition?");
    printf("Timebase position: %s\n", str_result);

    /* Set the acquisition type (NORMal, PEAK, or AVERage). */
    do_command(":ACQuire:TYPE NORMal");
    do_query_string(":ACQuire:TYPE?");
    printf("Acquire type: %s\n", str_result);

    /* Or, configure by loading a previously saved setup. */

    /* Read setup string from file. */
    fp = fopen("c:\\scope\\config\\setup.scp", "rb");
    num_bytes = fread(ieeeblock_data, sizeof(unsigned char),
        IEEEBLOCK_SPACE, fp);
    fclose(fp);
    printf("Read setup string (%d bytes) from file ", num_bytes);
    printf("c:\\scope\\config\\setup.stp.\n");

    /* Restore setup string. */
    num_bytes = do_command_ieeeblock(":SYSTem:SETup", num_bytes);
    printf("Restored setup string (%d bytes).\n", num_bytes);

    /* Capture an acquisition using :DIGitize. */
    do_command(":DIGitize CHANnel1");
}

/* Analyze the captured waveform.
 * ------------------------------------------------------------------- */
void analyze(void)
{
    double wav_format;
    double acq_type;
    double wav_points;
    double avg_count;
    double x_increment;
```

```
double x_origin;
double x_reference;
double y_increment;
double y_origin;
double y_reference;

FILE* fp;
int num_bytes;   /* Number of bytes returned from instrument. */
int i;

/* Make a couple of measurements.
 * ---------------------------------------------------------- */
do_command(":MEASure:SOURce CHANnel1");
do_query_string(":MEASure:SOURce?");
printf("Measure source: %s\n", str_result);

do_command(":MEASure:FREQuency");
do_query_number(":MEASure:FREQuency?");
printf("Frequency: %.4f kHz\n", num_result / 1000);

do_command(":MEASure:VAMPlitude");
do_query_number(":MEASure:VAMPlitude?");
printf("Vertical amplitude: %.2f V\n", num_result);

/* Download the screen image.
 * ---------------------------------------------------------- */

 /* Read screen image. */
num_bytes = do_query_ieeeblock(":DISPlay:DATA? PNG");
printf("Screen image bytes: %d\n", num_bytes);

/* Write screen image bytes to file. */
fp = fopen("c:\\scope\\data\\screen.png", "wb");
num_bytes = fwrite(ieeeblock_data, sizeof(unsigned char), num_bytes,
    fp);
fclose(fp);
printf("Wrote screen image (%d bytes) to ", num_bytes);
printf("c:\\scope\\data\\screen.png.\n");

/* Download waveform data.
 * ---------------------------------------------------------- */

 /* Set the waveform points mode. */
do_command(":WAVeform:POINts:MODE RAW");
do_query_string(":WAVeform:POINts:MODE?");
printf("Waveform points mode: %s\n", str_result);

/* Get the number of waveform points available. */
do_query_string(":WAVeform:POINts?");
printf("Waveform points available: %s\n", str_result);

/* Set the waveform source. */
do_command(":WAVeform:SOURce CHANnel1");
do_query_string(":WAVeform:SOURce?");
printf("Waveform source: %s\n", str_result);

/* Choose the format of the data returned (WORD, BYTE, ASCII): */
```

```
do_command(":WAVeform:FORMat WORD");
do_query_string(":WAVeform:FORMat?");
printf("Waveform format: %s\n", str_result);

/* Specify the byte order in the word format data: */
do_command(":WAVeform:BYTeorder LSBF");
do_query_string(":WAVeform:BYTeorder?");
printf("Waveform byte order: %s\n", str_result);

/* Display the waveform settings: */
do_query_numbers(":WAVeform:PREamble?");

wav_format = dbl_results[0];
if (wav_format == 0.0)
{
    printf("Waveform format: BYTE\n");
}
else if (wav_format == 1.0)
{
    printf("Waveform format: WORD\n");
}
else if (wav_format == 2.0)
{
    printf("Waveform format: ASCii\n");
}

acq_type = dbl_results[1];
if (acq_type == 0.0)
{
    printf("Acquire type: NORMal\n");
}
else if (acq_type == 1.0)
{
    printf("Acquire type: PEAK\n");
}
else if (acq_type == 2.0)
{
    printf("Acquire type: AVERage\n");
}

wav_points = dbl_results[2];
printf("Waveform points: %e\n", wav_points);

avg_count = dbl_results[3];
printf("Waveform average count: %e\n", avg_count);

x_increment = dbl_results[4];
printf("Waveform X increment: %e\n", x_increment);

x_origin = dbl_results[5];
printf("Waveform X origin: %e\n", x_origin);

x_reference = dbl_results[6];
printf("Waveform X reference: %e\n", x_reference);

y_increment = dbl_results[7];
printf("Waveform Y increment: %e\n", y_increment);
```

```
    y_origin = dbl_results[8];
    printf("Waveform Y origin: %e\n", y_origin);

    y_reference = dbl_results[9];
    printf("Waveform Y reference: %e\n", y_reference);

    /* Read waveform data. */
    num_bytes = do_query_ieeeblock(":WAVeform:DATA?");
    printf("Number of data values: %d\n", num_bytes / 2);

    /* Open file for output. */
    fp = fopen("c:\\scope\\data\\waveform_data.csv", "wb");

    /* Output waveform data in CSV format. */
    for (i = 0; i < num_bytes - 1; i+=2)
    {
        /* Write time value, voltage value. */
        fprintf(fp, "%9f, %6f\n",
            x_origin + ((float)(i / 2) * x_increment),
            (((float)(ieeeblock_data[i] + (ieeeblock_data[i+1] * 256))
                - y_reference) * y_increment) + y_origin);
    }

    /* Close output file. */
    fclose(fp);
    printf("Waveform format WORD data written to ");
    printf("c:\\scope\\data\\waveform_data.csv.\n");
}

/* Send a command to the instrument.
 * ---------------------------------------------------------------- */
void do_command(command)
char* command;
{
    char message[80];

    strcpy(message, command);
    strcat(message, "\n");
    err = viPrintf(vi, message);
    if (err != VI_SUCCESS) error_handler();

    check_instrument_errors();
}

/* Command with IEEE definite-length block.
 * ---------------------------------------------------------------- */
int do_command_ieeeblock(command, num_bytes)
char* command;
int num_bytes;
{
    char message[80];
    int data_length;

    strcpy(message, command);
    strcat(message, " #8%08d");
    err = viPrintf(vi, message, num_bytes);
```

```
    if (err != VI_SUCCESS) error_handler();

    err = viBufWrite(vi, ieeeblock_data, num_bytes, &data_length);
    if (err != VI_SUCCESS) error_handler();

    check_instrument_errors();

    return(data_length);
}

/* Query for a string result.
 * ------------------------------------------------------------- */
void do_query_string(query)
char* query;
{
    char message[80];

    strcpy(message, query);
    strcat(message, "\n");

    err = viPrintf(vi, message);
    if (err != VI_SUCCESS) error_handler();

    err = viScanf(vi, "%t", str_result);
    if (err != VI_SUCCESS) error_handler();

    check_instrument_errors();
}

/* Query for a number result.
 * ------------------------------------------------------------- */
void do_query_number(query)
char* query;
{
    char message[80];

    strcpy(message, query);
    strcat(message, "\n");

    err = viPrintf(vi, message);
    if (err != VI_SUCCESS) error_handler();

    err = viScanf(vi, "%lf", &num_result);
    if (err != VI_SUCCESS) error_handler();

    check_instrument_errors();
}

/* Query for numbers result.
 * ------------------------------------------------------------- */
void do_query_numbers(query)
char* query;
{
    char message[80];

    strcpy(message, query);
    strcat(message, "\n");
```

```
    err = viPrintf(vi, message);
    if (err != VI_SUCCESS) error_handler();

    err = viScanf(vi, "%,10lf\n", dbl_results);
    if (err != VI_SUCCESS) error_handler();

    check_instrument_errors();
}

/* Query for an IEEE definite-length block result.
 * ------------------------------------------------------------- */
int do_query_ieeeblock(query)
char* query;
{
    char message[80];
    int data_length;

    strcpy(message, query);
    strcat(message, "\n");
    err = viPrintf(vi, message);
    if (err != VI_SUCCESS) error_handler();

    data_length = IEEEBLOCK_SPACE;
    err = viScanf(vi, "%#b\n", &data_length, ieeeblock_data);
    if (err != VI_SUCCESS) error_handler();

    if (data_length == IEEEBLOCK_SPACE)
    {
        printf("IEEE block buffer full: ");
        printf("May not have received all data.\n");
    }

    check_instrument_errors();

    return(data_length);
}

/* Check for instrument errors.
 * ------------------------------------------------------------- */
void check_instrument_errors()
{
    char str_err_val[256] = { 0 };
    char str_out[800] = "";

    err = viQueryf(vi, ":SYSTem:ERRor?\n", "%t", str_err_val);
    if (err != VI_SUCCESS) error_handler();
    while (strncmp(str_err_val, "+0,No error", 3) != 0)
    {
        strcat(str_out, ", ");
        strcat(str_out, str_err_val);
        err = viQueryf(vi, ":SYSTem:ERRor?\n", "%t", str_err_val);
        if (err != VI_SUCCESS) error_handler();
    }

    if (strcmp(str_out, "") != 0)
    {
```

```
        printf("INST Error%s\n", str_out);
        err = viFlush(vi, VI_READ_BUF);
        if (err != VI_SUCCESS) error_handler();
        err = viFlush(vi, VI_WRITE_BUF);
        if (err != VI_SUCCESS) error_handler();
    }
}

/* Handle VISA errors.
 * ------------------------------------------------------------------- */
void error_handler()
{
    char err_msg[1024] = { 0 };

    viStatusDesc(vi, err, err_msg);
    printf("VISA Error: %s\n", err_msg);
    if (err < VI_SUCCESS)
    {
        exit(1);
    }
}
```

VISA Example in C#

To compile and run this example in Microsoft Visual Studio 2019:

1. Open Visual Studio.
2. Create a new Visual C#, Windows, Console Application project using the **.NET Framework 4.7.2**.
3. Cut-and-paste the code that follows into the C# source file.
4. Edit the program to use the VISA address of your oscilloscope.
5. Add Keysight's VISA header file to your project:
   a. Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment.
   b. Choose **Add > Existing Item...**
   c. Navigate to the header file, visa32.cs (installed with Keysight IO Libraries Suite and found in the Program Files (x86)\IVI Foundation\VISA\WinNT\ Include directory), select it, but *do not click the Open button*.
   d. Click the down arrow to the right of the **Add** button, and choose **Add as Link**.

      You should now see the file underneath your project in the Solution Explorer. It will have a little arrow icon in its lower left corner, indicating that it is a link.
6. Build and run the program.

For more information, see the tutorial on using VISA in Microsoft .NET in the VISA Help that comes with Keysight IO Libraries Suite.

```
/*
 * Keysight VISA Example in C#
 * ----------------------------------------------------------------------
 * This program illustrates a few commonly used programming
 * features of your Keysight oscilloscope.
 * ----------------------------------------------------------------------
 */

using System;
using System.IO;
using System.Text;

namespace InfiniiVision
{
  class VisaInstrumentApp
  {
    private static VisaInstrument myScope;

    public static void Main(string[] args)
    {
      try
      {
        myScope = new
          VisaInstrument("USB0::0x2A8D::0x4704::Unset::0::INSTR");
        myScope.SetTimeoutSeconds(10);

        // Initialize - start from a known state.
        Initialize();

        // Capture data.
        Capture();

        // Analyze the captured waveform.
        Analyze();

      }
      catch (System.ApplicationException err)
      {
        Console.WriteLine("*** VISA Error Message : " + err.Message);
      }
      catch (System.SystemException err)
      {
        Console.WriteLine("*** System Error Message : " + err.Message);
      }
      catch (System.Exception err)
      {
        System.Diagnostics.Debug.Fail("Unexpected Error");
        Console.WriteLine("*** Unexpected Error : " + err.Message);
      }
      finally
      {
        myScope.Close();
      }
    }

    /*
     * Initialize the oscilloscope to a known state.
```

```
 * -------------------------------------------------------------
 */
private static void Initialize()
{
  StringBuilder strResults;

  // Get and display the device's *IDN? string.
  strResults = myScope.DoQueryString("*IDN?");
  Console.WriteLine("*IDN? result is: {0}", strResults);

  // Clear status and load the default setup.
  myScope.DoCommand("*CLS");
  myScope.DoCommand("*RST");
}

/*
 * Capture the waveform.
 * -------------------------------------------------------------
 */
private static void Capture()
{
  // Use auto-scale to automatically configure oscilloscope.
  myScope.DoCommand(":AUToscale");

  // Set trigger mode (EDGE, PULSe, PATTern, etc., and input source.
  myScope.DoCommand(":TRIGger:MODE EDGE");
  Console.WriteLine("Trigger mode: {0}",
      myScope.DoQueryString(":TRIGger:MODE?"));

  // Set EDGE trigger parameters.
  myScope.DoCommand(":TRIGger:EDGE:SOURce CHANnel1");
  Console.WriteLine("Trigger edge source: {0}",
      myScope.DoQueryString(":TRIGger:EDGE:SOURce?"));

  myScope.DoCommand(":TRIGger:EDGE:LEVel 1.5");
  Console.WriteLine("Trigger edge level: {0}",
      myScope.DoQueryString(":TRIGger:EDGE:LEVel?"));

  myScope.DoCommand(":TRIGger:EDGE:SLOPe POSitive");
  Console.WriteLine("Trigger edge slope: {0}",
      myScope.DoQueryString(":TRIGger:EDGE:SLOPe?"));

  // Save oscilloscope configuration.
  byte[] ResultsArray;   // Results array.
  int nLength;   // Number of bytes returned from instrument.
  string strPath;

  // Query and read setup string.
  nLength = myScope.DoQueryIEEEBlock(":SYSTem:SETup?",
    out ResultsArray);

  // Write setup string to file.
  strPath = "c:\\scope\\config\\setup.scp";
  FileStream fStream = File.Open(strPath, FileMode.Create);
  fStream.Write(ResultsArray, 0, nLength);
  fStream.Close();
  Console.WriteLine("Setup bytes saved: {0}", nLength);
```

```
  // Change settings with individual commands:

  // Set vertical scale and offset.
  myScope.DoCommand(":CHANnel1:SCALe 0.05");
  Console.WriteLine("Channel 1 vertical scale: {0}",
      myScope.DoQueryString(":CHANnel1:SCALe?"));

  myScope.DoCommand(":CHANnel1:OFFSet -1.5");
  Console.WriteLine("Channel 1 vertical offset: {0}",
      myScope.DoQueryString(":CHANnel1:OFFSet?"));

  // Set horizontal scale and position.
  myScope.DoCommand(":TIMebase:SCALe 0.0002");
  Console.WriteLine("Timebase scale: {0}",
      myScope.DoQueryString(":TIMebase:SCALe?"));

  myScope.DoCommand(":TIMebase:POSition 0.0");
  Console.WriteLine("Timebase position: {0}",
      myScope.DoQueryString(":TIMebase:POSition?"));

  // Set the acquisition type (NORMal, PEAK, or AVERage).
  myScope.DoCommand(":ACQuire:TYPE NORMal");
  Console.WriteLine("Acquire type: {0}",
      myScope.DoQueryString(":ACQuire:TYPE?"));

  // Or, configure by loading a previously saved setup.
  byte[] DataArray;
  int nBytesWritten;

  // Read setup string from file.
  strPath = "c:\\scope\\config\\setup.scp";
  DataArray = File.ReadAllBytes(strPath);

  // Restore setup string.
  nBytesWritten = myScope.DoCommandIEEEBlock(":SYSTem:SETup",
    DataArray);
  Console.WriteLine("Setup bytes restored: {0}", nBytesWritten);

  // Capture an acquisition using :DIGitize.
  myScope.DoCommand(":DIGitize CHANnel1");
}

/*
 * Analyze the captured waveform.
 * -------------------------------------------------------------------
 */
private static void Analyze()
{
  byte[] ResultsArray;   // Results array.
  int nLength;   // Number of bytes returned from instrument.
  string strPath;

  // Make a couple of measurements.
  // ----------------------------------------------------------
  myScope.DoCommand(":MEASure:SOURce CHANnel1");
  Console.WriteLine("Measure source: {0}",
```

```
    myScope.DoQueryString(":MEASure:SOURce?"));

double fResult;
myScope.DoCommand(":MEASure:FREQuency");
fResult = myScope.DoQueryNumber(":MEASure:FREQuency?");
Console.WriteLine("Frequency: {0:F4} kHz", fResult / 1000);

myScope.DoCommand(":MEASure:VAMPlitude");
fResult = myScope.DoQueryNumber(":MEASure:VAMPlitude?");
Console.WriteLine("Vertical amplitude: {0:F2} V", fResult);

// Download the screen image.
// ------------------------------------------------------------

// Get the screen data.
nLength = myScope.DoQueryIEEEBlock(":DISPlay:DATA? PNG",
    out ResultsArray);

// Store the screen data to a file.
strPath = "c:\\scope\\data\\screen.png";
FileStream fStream = File.Open(strPath, FileMode.Create);
fStream.Write(ResultsArray, 0, nLength);
fStream.Close();
Console.WriteLine("Screen image ({0} bytes) written to {1}",
    nLength, strPath);

// Download waveform data.
// ------------------------------------------------------------

// Set the waveform points mode.
myScope.DoCommand(":WAVeform:POINts:MODE RAW");
Console.WriteLine("Waveform points mode: {0}",
    myScope.DoQueryString(":WAVeform:POINts:MODE?"));

// Get the number of waveform points available.
myScope.DoCommand(":WAVeform:POINts 10240");
Console.WriteLine("Waveform points available: {0}",
    myScope.DoQueryString(":WAVeform:POINts?"));

// Set the waveform source.
myScope.DoCommand(":WAVeform:SOURce CHANnel1");
Console.WriteLine("Waveform source: {0}",
    myScope.DoQueryString(":WAVeform:SOURce?"));

// Choose the format of the data returned (WORD, BYTE, ASCII):
myScope.DoCommand(":WAVeform:FORMat WORD");
Console.WriteLine("Waveform format: {0}",
    myScope.DoQueryString(":WAVeform:FORMat?"));

// Specify the byte order in the word format data:
myScope.DoCommand(":WAVeform:BYTeorder LSBF");
Console.WriteLine("Waveform byte order: {0}",
    myScope.DoQueryString(":WAVeform:BYTeorder?"));

// Display the waveform settings:
double[] fResultsArray;
fResultsArray = myScope.DoQueryNumbers(":WAVeform:PREamble?");
```

```
double fFormat = fResultsArray[0];
if (fFormat == 0.0)
{
  Console.WriteLine("Waveform format: BYTE");
}
else if (fFormat == 1.0)
{
  Console.WriteLine("Waveform format: WORD");
}
else if (fFormat == 2.0)
{
  Console.WriteLine("Waveform format: ASCii");
}

double fType = fResultsArray[1];
if (fType == 0.0)
{
  Console.WriteLine("Acquire type: NORMal");
}
else if (fType == 1.0)
{
  Console.WriteLine("Acquire type: PEAK");
}
else if (fType == 2.0)
{
  Console.WriteLine("Acquire type: AVERage");
}

double fPoints = fResultsArray[2];
Console.WriteLine("Waveform points: {0:e}", fPoints);

double fCount = fResultsArray[3];
Console.WriteLine("Waveform average count: {0:e}", fCount);

double fXincrement = fResultsArray[4];
Console.WriteLine("Waveform X increment: {0:e}", fXincrement);

double fXorigin = fResultsArray[5];
Console.WriteLine("Waveform X origin: {0:e}", fXorigin);

double fXreference = fResultsArray[6];
Console.WriteLine("Waveform X reference: {0:e}", fXreference);

double fYincrement = fResultsArray[7];
Console.WriteLine("Waveform Y increment: {0:e}", fYincrement);

double fYorigin = fResultsArray[8];
Console.WriteLine("Waveform Y origin: {0:e}", fYorigin);

double fYreference = fResultsArray[9];
Console.WriteLine("Waveform Y reference: {0:e}", fYreference);

// Read waveform data.
nLength = myScope.DoQueryIEEEBlock(":WAVeform:DATA?",
    out ResultsArray);
Console.WriteLine("Number of data values: {0}", nLength / 2);
```

```
      // Set up output file:
      strPath = "c:\\scope\\data\\waveform_data.csv";
      if (File.Exists(strPath)) File.Delete(strPath);

      // Open file for output.
      StreamWriter writer = File.CreateText(strPath);

      // Output waveform data in CSV format.
      for (int i = 0; i < nLength - 1; i+=2)
        writer.WriteLine("{0:f9}, {1:f6}",
            fXorigin + ((float)(i / 2) * fXincrement),
            (((float)BitConverter.ToUInt16(ResultsArray, i) - fYreferenc
e) *
            fYincrement) + fYorigin);

      // Close output file.
      writer.Close();
      Console.WriteLine("Waveform format WORD data written to {0}",
          strPath);
    }
  }

  class VisaInstrument
  {
    private int m_nResourceManager;
    private int m_nSession;
    private string m_strVisaAddress;

    // Constructor.
    public VisaInstrument(string strVisaAddress)
    {
      // Save VISA address in member variable.
      m_strVisaAddress = strVisaAddress;

      // Open the default VISA resource manager.
      OpenResourceManager();

      // Open a VISA resource session.
      OpenSession();

      // Clear the interface.
      int nViStatus;
      nViStatus = visa32.viClear(m_nSession);
    }

    public void DoCommand(string strCommand)
    {
      // Send the command.
      VisaSendCommandOrQuery(strCommand);

      // Check for inst errors.
      CheckInstrumentErrors(strCommand);
    }

    public int DoCommandIEEEBlock(string strCommand,
      byte[] DataArray)
```

```
{
  // Send the command to the device.
  string strCommandAndLength;
  int nViStatus, nLength, nBytesWritten;

  nLength = DataArray.Length;
  strCommandAndLength = String.Format("{0} #8%08d",
    strCommand);

  // Write first part of command to formatted I/O write buffer.
  nViStatus = visa32.viPrintf(m_nSession, strCommandAndLength,
    nLength);
  CheckVisaStatus(nViStatus);

  // Write the data to the formatted I/O write buffer.
  nViStatus = visa32.viBufWrite(m_nSession, DataArray, nLength,
    out nBytesWritten);
  CheckVisaStatus(nViStatus);

  // Check for inst errors.
  CheckInstrumentErrors(strCommand);

  return nBytesWritten;
}

public StringBuilder DoQueryString(string strQuery)
{
  // Send the query.
  VisaSendCommandOrQuery(strQuery);

  // Get the result string.
  StringBuilder strResults = new StringBuilder(1000);
  strResults = VisaGetResultString();

  // Check for inst errors.
  CheckInstrumentErrors(strQuery);

  // Return string results.
  return strResults;
}

public double DoQueryNumber(string strQuery)
{
  // Send the query.
  VisaSendCommandOrQuery(strQuery);

  // Get the result string.
  double fResults;
  fResults = VisaGetResultNumber();

  // Check for inst errors.
  CheckInstrumentErrors(strQuery);

  // Return string results.
  return fResults;
}
```

```
public double[] DoQueryNumbers(string strQuery)
{
  // Send the query.
  VisaSendCommandOrQuery(strQuery);

  // Get the result string.
  double[] fResultsArray;
  fResultsArray = VisaGetResultNumbers();

  // Check for inst errors.
  CheckInstrumentErrors(strQuery);

  // Return string results.
  return fResultsArray;
}

public int DoQueryIEEEBlock(string strQuery,
  out byte[] ResultsArray)
{
  // Send the query.
  VisaSendCommandOrQuery(strQuery);

  // Get the result string.
  int length;   // Number of bytes returned from instrument.
  length = VisaGetResultIEEEBlock(out ResultsArray);

  // Check for inst errors.
  CheckInstrumentErrors(strQuery);

  // Return string results.
  return length;
}

private void VisaSendCommandOrQuery(string strCommandOrQuery)
{
  // Send command or query to the device.
  string strWithNewline;
  strWithNewline = String.Format("{0}\n", strCommandOrQuery);
  int nViStatus;
  nViStatus = visa32.viPrintf(m_nSession, strWithNewline);
  CheckVisaStatus(nViStatus);
}

private StringBuilder VisaGetResultString()
{
  StringBuilder strResults = new StringBuilder(1000);

  // Read return value string from the device.
  int nViStatus;
  nViStatus = visa32.viScanf(m_nSession, "%1000t", strResults);
  CheckVisaStatus(nViStatus);

  return strResults;
}

private double VisaGetResultNumber()
{
```

```
  double fResults = 0;

  // Read return value string from the device.
  int nViStatus;
  nViStatus = visa32.viScanf(m_nSession, "%lf", out fResults);
  CheckVisaStatus(nViStatus);

  return fResults;
}

private double[] VisaGetResultNumbers()
{
  double[] fResultsArray;
  fResultsArray = new double[10];

  // Read return value string from the device.
  int nViStatus;
  nViStatus = visa32.viScanf(m_nSession, "%,10lf\n",
      fResultsArray);
  CheckVisaStatus(nViStatus);

  return fResultsArray;
}

private int VisaGetResultIEEEBlock(out byte[] ResultsArray)
{
  // Results array, big enough to hold a PNG.
  ResultsArray = new byte[3000000];
  int length;   // Number of bytes returned from instrument.

  // Set the default number of bytes that will be contained in
  // the ResultsArray to 300,000 (300kB).
  length = 3000000;

  // Read return value string from the device.
  int nViStatus;
  nViStatus = visa32.viScanf(m_nSession, "%#b", ref length,
    ResultsArray);
  CheckVisaStatus(nViStatus);

  // Write and read buffers need to be flushed after IEEE block?
  nViStatus = visa32.viFlush(m_nSession, visa32.VI_WRITE_BUF);
  CheckVisaStatus(nViStatus);

  nViStatus = visa32.viFlush(m_nSession, visa32.VI_READ_BUF);
  CheckVisaStatus(nViStatus);

  return length;
}

private void CheckInstrumentErrors(string strCommand)
{
  // Check for instrument errors.
  StringBuilder strInstrumentError = new StringBuilder(1000);
  bool bFirstError = true;

  do   // While not "0,No error"
```

```
  {
    VisaSendCommandOrQuery(":SYSTem:ERRor?");
    strInstrumentError = VisaGetResultString();

    if (!strInstrumentError.ToString().StartsWith("+0,"))
    {
      if (bFirstError)
      {
        Console.WriteLine("ERROR(s) for command '{0}': ",
          strCommand);
        bFirstError = false;
      }
      Console.Write(strInstrumentError);
    }
  } while (!strInstrumentError.ToString().StartsWith("+0,"));
}

private void OpenResourceManager()
{
  int nViStatus;
  nViStatus =
    visa32.viOpenDefaultRM(out this.m_nResourceManager);
  if (nViStatus < visa32.VI_SUCCESS)
    throw new
      ApplicationException("Failed to open Resource Manager");
}

private void OpenSession()
{
  int nViStatus;
  nViStatus = visa32.viOpen(this.m_nResourceManager,
    this.m_strVisaAddress, visa32.VI_NO_LOCK,
    visa32.VI_TMO_IMMEDIATE, out this.m_nSession);
  CheckVisaStatus(nViStatus);
}

public void SetTimeoutSeconds(int nSeconds)
{
  int nViStatus;
  nViStatus = visa32.viSetAttribute(this.m_nSession,
    visa32.VI_ATTR_TMO_VALUE, nSeconds * 1000);
  CheckVisaStatus(nViStatus);
}

public void CheckVisaStatus(int nViStatus)
{
  // If VISA error, throw exception.
  if (nViStatus < visa32.VI_SUCCESS)
  {
    StringBuilder strError = new StringBuilder(256);
    visa32.viStatusDesc(this.m_nResourceManager, nViStatus,
      strError);
    throw new ApplicationException(strError.ToString());
  }
}

public void Close()
```

```
    {
      if (m_nSession != 0)
        visa32.viClose(m_nSession);
      if (m_nResourceManager != 0)
        visa32.viClose(m_nResourceManager);
    }
  }
}
```

VISA Example in Visual Basic .NET

To compile and run this example in Microsoft Visual Studio 2019:

1. Open Visual Studio.
2. Create a new Visual Basic, Windows, Console Application project using the **.NET Framework 4.7.2**.
3. Cut-and-paste the code that follows into the Visual Basic .NET source file.
4. Edit the program to use the VISA address of your oscilloscope.
5. Add Keysight's VISA header file to your project:
   a. Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment.
   b. Choose **Add > Existing Item...**
   c. Navigate to the header file, visa32.vb (installed with Keysight IO Libraries Suite and found in the Program Files (x86)\IVI Foundation\VISA\WinNT\Include directory), select it, but *do not click the Open button*.
   d. Click the down arrow to the right of the **Add** button, and choose **Add as Link**.

      You should now see the file underneath your project in the Solution Explorer. It will have a little arrow icon in its lower left corner, indicating that it is a link.
   e. Right-click the project again and choose **Properties**; then, select "InfiniiVision.VisaInstrumentApp" as the **Startup object**.
6. Build and run the program.

For more information, see the tutorial on using VISA in Microsoft .NET in the VISA Help that comes with Keysight IO Libraries Suite.

```
'
' Keysight VISA Example in Visual Basic .NET
' ----------------------------------------------------------------------
' This program illustrates a few commonly-used programming
' features of your Keysight oscilloscope.
' ----------------------------------------------------------------------

Imports System
Imports System.IO
Imports System.Text

Namespace InfiniiVision
```

```
Class VisaInstrumentApp
  Private Shared myScope As VisaInstrument

  Public Shared Sub Main(ByVal args As String())
    Try
      myScope =
        New VisaInstrument("USB0::0x2A8D::0x4704::Unset::0::INSTR")
      myScope.SetTimeoutSeconds(10)

      ' Initialize - start from a known state.
      Initialize()

      ' Capture data.
      Capture()

      ' Analyze the captured waveform.
      Analyze()

    Catch err As System.ApplicationException
      Console.WriteLine("*** VISA Error Message : " + err.Message)
    Catch err As System.SystemException
      Console.WriteLine("*** System Error Message : " + err.Message)
    Catch err As System.Exception
      Debug.Fail("Unexpected Error")
      Console.WriteLine("*** Unexpected Error : " + err.Message)
    End Try
  End Sub

  '
  ' Initialize the oscilloscope to a known state.
  ' ---------------------------------------------------------------

  Private Shared Sub Initialize()
    Dim strResults As StringBuilder

    ' Get and display the device's *IDN? string.
    strResults = myScope.DoQueryString("*IDN?")
    Console.WriteLine("*IDN? result is: {0}", strResults)

    ' Clear status and load the default setup.
    myScope.DoCommand("*CLS")
    myScope.DoCommand("*RST")

  End Sub

  '
  ' Capture the waveform.
  ' ---------------------------------------------------------------

  Private Shared Sub Capture()

    ' Use auto-scale to automatically configure oscilloscope.
    myScope.DoCommand(":AUToscale")

    ' Set trigger mode (EDGE, PULSe, PATTern, etc., and input source.
    myScope.DoCommand(":TRIGger:MODE EDGE")
    Console.WriteLine("Trigger mode: {0}",
```

```
    myScope.DoQueryString(":TRIGger:MODE?"))

' Set EDGE trigger parameters.
myScope.DoCommand(":TRIGger:EDGE:SOURce CHANnel1")
Console.WriteLine("Trigger edge source: {0}",
    myScope.DoQueryString(":TRIGger:EDGE:SOURce?"))

myScope.DoCommand(":TRIGger:EDGE:LEVel 1.5")
Console.WriteLine("Trigger edge level: {0}",
    myScope.DoQueryString(":TRIGger:EDGE:LEVel?"))

myScope.DoCommand(":TRIGger:EDGE:SLOPe POSitive")
Console.WriteLine("Trigger edge slope: {0}",
    myScope.DoQueryString(":TRIGger:EDGE:SLOPe?"))

' Save oscilloscope configuration.
Dim ResultsArray As Byte()   ' Results array.
Dim nLength As Integer   ' Number of bytes returned from inst.
Dim strPath As String
Dim fStream As FileStream

' Query and read setup string.
nLength = myScope.DoQueryIEEEBlock(":SYSTem:SETup?",
    ResultsArray)

' Write setup string to file.
strPath = "c:\scope\config\setup.scp"
fStream = File.Open(strPath, FileMode.Create)
fStream.Write(ResultsArray, 0, nLength)
fStream.Close()
Console.WriteLine("Setup bytes saved: {0}", nLength)

' Change settings with individual commands:

' Set vertical scale and offset.
myScope.DoCommand(":CHANnel1:SCALe 0.05")
Console.WriteLine("Channel 1 vertical scale: {0}",
    myScope.DoQueryString(":CHANnel1:SCALe?"))

myScope.DoCommand(":CHANnel1:OFFSet -1.5")
Console.WriteLine("Channel 1 vertical offset: {0}",
    myScope.DoQueryString(":CHANnel1:OFFSet?"))

' Set horizontal scale and position.
myScope.DoCommand(":TIMebase:SCALe 0.0002")
Console.WriteLine("Timebase scale: {0}",
    myScope.DoQueryString(":TIMebase:SCALe?"))

myScope.DoCommand(":TIMebase:POSition 0.0")
Console.WriteLine("Timebase position: {0}",
    myScope.DoQueryString(":TIMebase:POSition?"))

' Set the acquisition type (NORMal, PEAK, or AVERage).
myScope.DoCommand(":ACQuire:TYPE NORMal")
Console.WriteLine("Acquire type: {0}",
    myScope.DoQueryString(":ACQuire:TYPE?"))
```

```
  ' Or, configure by loading a previously saved setup.
  Dim DataArray As Byte()
  Dim nBytesWritten As Integer

  ' Read setup string from file.
  strPath = "c:\scope\config\setup.scp"
  DataArray = File.ReadAllBytes(strPath)

  ' Restore setup string.
  nBytesWritten = myScope.DoCommandIEEEBlock(":SYSTem:SETup",
      DataArray)
  Console.WriteLine("Setup bytes restored: {0}", nBytesWritten)

  ' Capture an acquisition using :DIGitize.
  myScope.DoCommand(":DIGitize CHANnel1")

End Sub

'
' Analyze the captured waveform.
' ---------------------------------------------------------------

Private Shared Sub Analyze()

  Dim fResult As Double
  Dim ResultsArray As Byte()   ' Results array.
  Dim nLength As Integer   ' Number of bytes returned from inst.
  Dim strPath As String

  ' Make a couple of measurements.
  ' ---------------------------------------------------------------
  myScope.DoCommand(":MEASure:SOURce CHANnel1")
  Console.WriteLine("Measure source: {0}",
      myScope.DoQueryString(":MEASure:SOURce?"))

  myScope.DoCommand(":MEASure:FREQuency")
  fResult = myScope.DoQueryNumber(":MEASure:FREQuency?")
  Console.WriteLine("Frequency: {0:F4} kHz", fResult / 1000)

  myScope.DoCommand(":MEASure:VAMPlitude")
  fResult = myScope.DoQueryNumber(":MEASure:VAMPlitude?")
  Console.WriteLine("Vertical amplitude: {0:F2} V", fResult)

  ' Download the screen image.
  ' ---------------------------------------------------------------

  ' Get the screen data.
  nLength = myScope.DoQueryIEEEBlock(":DISPlay:DATA? PNG",
      ResultsArray)

  ' Store the screen data to a file.
  strPath = "c:\scope\data\screen.png"
  Dim fStream As FileStream
  fStream = File.Open(strPath, FileMode.Create)
  fStream.Write(ResultsArray, 0, nLength)
  fStream.Close()
  Console.WriteLine("Screen image ({0} bytes) written to {1}",
```

```
    nLength, strPath)

' Download waveform data.
' ------------------------------------------------------------

' Set the waveform points mode.
myScope.DoCommand(":WAVeform:POINts:MODE RAW")
Console.WriteLine("Waveform points mode: {0}",
    myScope.DoQueryString(":WAVeform:POINts:MODE?"))

' Get the number of waveform points available.
myScope.DoCommand(":WAVeform:POINts 10240")
Console.WriteLine("Waveform points available: {0}",
    myScope.DoQueryString(":WAVeform:POINts?"))

' Set the waveform source.
myScope.DoCommand(":WAVeform:SOURce CHANnel1")
Console.WriteLine("Waveform source: {0}",
    myScope.DoQueryString(":WAVeform:SOURce?"))

' Choose the format of the data returned (WORD, BYTE, ASCII):
myScope.DoCommand(":WAVeform:FORMat WORD")
Console.WriteLine("Waveform format: {0}",
    myScope.DoQueryString(":WAVeform:FORMat?"))

' Specify the byte order in the word format data:
myScope.DoCommand(":WAVeform:BYTeorder LSBF")
Console.WriteLine("Waveform byte order: {0}",
    myScope.DoQueryString(":WAVeform:BYTeorder?"))

' Display the waveform settings:
Dim fResultsArray As Double()
fResultsArray = myScope.DoQueryNumbers(":WAVeform:PREamble?")

Dim fFormat As Double = fResultsArray(0)
If fFormat = 0 Then
  Console.WriteLine("Waveform format: BYTE")
ElseIf fFormat = 1 Then
  Console.WriteLine("Waveform format: WORD")
ElseIf fFormat = 2 Then
  Console.WriteLine("Waveform format: ASCii")
End If

Dim fType As Double = fResultsArray(1)
If fType = 0 Then
  Console.WriteLine("Acquire type: NORMal")
ElseIf fType = 1 Then
  Console.WriteLine("Acquire type: PEAK")
ElseIf fType = 2 Then
  Console.WriteLine("Acquire type: AVERage")
End If

Dim fPoints As Double = fResultsArray(2)
Console.WriteLine("Waveform points: {0:e}", fPoints)

Dim fCount As Double = fResultsArray(3)
Console.WriteLine("Waveform average count: {0:e}", fCount)
```

```
        Dim fXincrement As Double = fResultsArray(4)
        Console.WriteLine("Waveform X increment: {0:e}", fXincrement)

        Dim fXorigin As Double = fResultsArray(5)
        Console.WriteLine("Waveform X origin: {0:e}", fXorigin)

        Dim fXreference As Double = fResultsArray(6)
        Console.WriteLine("Waveform X reference: {0:e}", fXreference)

        Dim fYincrement As Double = fResultsArray(7)
        Console.WriteLine("Waveform Y increment: {0:e}", fYincrement)

        Dim fYorigin As Double = fResultsArray(8)
        Console.WriteLine("Waveform Y origin: {0:e}", fYorigin)

        Dim fYreference As Double = fResultsArray(9)
        Console.WriteLine("Waveform Y reference: {0:e}", fYreference)

        ' Get the waveform data.
        nLength = myScope.DoQueryIEEEBlock(":WAVeform:DATA?",
            ResultsArray)
        Console.WriteLine("Number of data values: {0}", nLength / 2)

        ' Set up output file:
        strPath = "c:\scope\data\waveform_data.csv"
        If File.Exists(strPath) Then
          File.Delete(strPath)
        End If

        ' Open file for output.
        Dim writer As StreamWriter = File.CreateText(strPath)

        ' Output waveform data in CSV format.
        For index As Integer = 0 To nLength - 1 Step 2
          ' Write time value, voltage value.
          writer.WriteLine("{0:f9}, {1:f6}",
              fXorigin + (CSng(index / 2) * fXincrement),
              ((CSng(BitConverter.ToUInt16(ResultsArray, index)) - fYrefer
ence) _
              * fYincrement) + fYorigin)
        Next

        ' Close output file.
        writer.Close()
        Console.WriteLine("Waveform format WORD data written to {0}",
            strPath)

      End Sub

    End Class

    Class VisaInstrument
      Private m_nResourceManager As Integer
      Private m_nSession As Integer
      Private m_strVisaAddress As String
```

```
' Constructor.
Public Sub New(ByVal strVisaAddress As String)
  ' Save VISA address in member variable.
  m_strVisaAddress = strVisaAddress

  ' Open the default VISA resource manager.
  OpenResourceManager()

  ' Open a VISA resource session.
  OpenSession()

  ' Clear the interface.
  Dim nViStatus As Integer
  nViStatus = visa32.viClear(m_nSession)
End Sub

Public Sub DoCommand(ByVal strCommand As String)
  ' Send the command.
  VisaSendCommandOrQuery(strCommand)

  ' Check for inst errors.
  CheckInstrumentErrors(strCommand)

End Sub

Public Function DoCommandIEEEBlock(ByVal strCommand As String,
    ByVal DataArray As Byte()) As Integer

  ' Send the command to the device.
  Dim strCommandAndLength As String
  Dim nViStatus As Integer
  Dim nLength As Integer
  Dim nBytesWritten As Integer

  nLength = DataArray.Length
  strCommandAndLength = [String].Format("{0} #8{1:D8}",
      strCommand, nLength)

  ' Write first part of command to formatted I/O write buffer.
  nViStatus = visa32.viPrintf(m_nSession, strCommandAndLength)
  CheckVisaStatus(nViStatus)

  ' Write the data to the formatted I/O write buffer.
  nViStatus = visa32.viBufWrite(m_nSession, DataArray, nLength,
      nBytesWritten)
  CheckVisaStatus(nViStatus)

  ' Check for inst errors.
  CheckInstrumentErrors(strCommand)

  Return nBytesWritten
End Function

Public Function DoQueryString(ByVal strQuery As String) _
  As StringBuilder
  ' Send the query.
  VisaSendCommandOrQuery(strQuery)
```

```
  ' Get the result string.
  Dim strResults As New StringBuilder(1000)
  strResults = VisaGetResultString()

  ' Check for inst errors.
  CheckInstrumentErrors(strQuery)

  ' Return string results.
  Return strResults
End Function

Public Function DoQueryNumber(ByVal strQuery As String) As Double
  ' Send the query.
  VisaSendCommandOrQuery(strQuery)

  ' Get the result string.
  Dim fResults As Double
  fResults = VisaGetResultNumber()

  ' Check for inst errors.
  CheckInstrumentErrors(strQuery)

  ' Return string results.
  Return fResults
End Function

Public Function DoQueryNumbers(ByVal strQuery As String) _
    As Double()
  ' Send the query.
  VisaSendCommandOrQuery(strQuery)

  ' Get the result string.
  Dim fResultsArray As Double()
  fResultsArray = VisaGetResultNumbers()

  ' Check for instrument errors (another command and result).
  CheckInstrumentErrors(strQuery)

  ' Return string results.
  Return fResultsArray
End Function

Public Function DoQueryIEEEBlock(ByVal strQuery As String,
    ByRef ResultsArray As Byte()) As Integer
  ' Send the query.
  VisaSendCommandOrQuery(strQuery)

  ' Get the result string.
  System.Threading.Thread.Sleep(2000) ' Delay before reading data.
  Dim length As Integer
  ' Number of bytes returned from instrument.
  length = VisaGetResultIEEEBlock(ResultsArray)

  ' Check for inst errors.
  CheckInstrumentErrors(strQuery)
```

```
  ' Return string results.
  Return length
End Function

Private Sub VisaSendCommandOrQuery(ByVal strCommandOrQuery _
     As String)
  ' Send command or query to the device.
  Dim strWithNewline As String
  strWithNewline = [String].Format("{0}" & Chr(10) & "",
      strCommandOrQuery)
  Dim nViStatus As Integer
  nViStatus = visa32.viPrintf(m_nSession, strWithNewline)
  CheckVisaStatus(nViStatus)
End Sub

Private Function VisaGetResultString() As StringBuilder
  Dim strResults As New StringBuilder(1000)

  ' Read return value string from the device.
  Dim nViStatus As Integer
  nViStatus = visa32.viScanf(m_nSession, "%1000t", strResults)
  CheckVisaStatus(nViStatus)

  Return strResults
End Function

Private Function VisaGetResultNumber() As Double
  Dim fResults As Double = 0

  ' Read return value string from the device.
  Dim nViStatus As Integer
  nViStatus = visa32.viScanf(m_nSession, "%lf", fResults)
  CheckVisaStatus(nViStatus)

  Return fResults
End Function

Private Function VisaGetResultNumbers() As Double()
  Dim fResultsArray As Double()
  fResultsArray = New Double(9) {}

  ' Read return value string from the device.
  Dim nViStatus As Integer
  nViStatus = visa32.viScanf(m_nSession,
      "%,10lf" & Chr(10) & "", fResultsArray)
  CheckVisaStatus(nViStatus)

  Return fResultsArray
End Function

Private Function VisaGetResultIEEEBlock(ByRef ResultsArray _
    As Byte()) As Integer
  ' Results array, big enough to hold a PNG.
  ResultsArray = New Byte(2999999) {}
  Dim length As Integer
  ' Number of bytes returned from instrument.
  ' Set the default number of bytes that will be contained in
```

```
  ' the ResultsArray to 3,000,000 (~3MB).
  length = 3000000

  ' Read return value string from the device.
  Dim nViStatus As Integer
  nViStatus = visa32.viScanf(m_nSession, "%#b", length,
      ResultsArray)
  CheckVisaStatus(nViStatus)

  ' Write and read buffers need to be flushed after IEEE block?
  nViStatus = visa32.viFlush(m_nSession, visa32.VI_WRITE_BUF)
  CheckVisaStatus(nViStatus)

  nViStatus = visa32.viFlush(m_nSession, visa32.VI_READ_BUF)
  CheckVisaStatus(nViStatus)

  Return length
End Function

Private Sub CheckInstrumentErrors(ByVal strCommand As String)
  ' Check for instrument errors.
  Dim strInstrumentError As New StringBuilder(1000)
  Dim bFirstError As Boolean = True
  Do   ' While not "0,No error"
    VisaSendCommandOrQuery(":SYSTem:ERRor?")
    strInstrumentError = VisaGetResultString()

    If Not strInstrumentError.ToString().StartsWith("+0,") Then
      If bFirstError Then
        Console.WriteLine("ERROR(s) for command '{0}': ",
            strCommand)
        bFirstError = False
      End If
      Console.Write(strInstrumentError)
    End If
  Loop While Not strInstrumentError.ToString().StartsWith("+0,")
End Sub

Private Sub OpenResourceManager()
  Dim nViStatus As Integer
  nViStatus = visa32.viOpenDefaultRM(Me.m_nResourceManager)
  If nViStatus < visa32.VI_SUCCESS Then
    Throw New _
        ApplicationException("Failed to open Resource Manager")
  End If
End Sub

Private Sub OpenSession()
  Dim nViStatus As Integer
  nViStatus = visa32.viOpen(Me.m_nResourceManager,
      Me.m_strVisaAddress, visa32.VI_NO_LOCK,
      visa32.VI_TMO_IMMEDIATE, Me.m_nSession)
  CheckVisaStatus(nViStatus)
End Sub

Public Sub SetTimeoutSeconds(ByVal nSeconds As Integer)
  Dim nViStatus As Integer
```

```
      nViStatus = visa32.viSetAttribute(Me.m_nSession,
          visa32.VI_ATTR_TMO_VALUE, nSeconds * 1000)
      CheckVisaStatus(nViStatus)
    End Sub

    Public Sub CheckVisaStatus(ByVal nViStatus As Integer)
      ' If VISA error, throw exception.
      If nViStatus < visa32.VI_SUCCESS Then
        Dim strError As New StringBuilder(256)
        visa32.viStatusDesc(Me.m_nResourceManager, nViStatus, strError)
        Throw New ApplicationException(strError.ToString())
      End If
    End Sub

    Public Sub Close()
      If m_nSession <> 0 Then
        visa32.viClose(m_nSession)
      End If
      If m_nResourceManager <> 0 Then
        visa32.viClose(m_nResourceManager)
      End If
    End Sub
  End Class
End Namespace
```

VISA Example in Python

You can use the Python programming language with the PyVISA package to control Keysight InfiniiVision Series oscilloscopes.

The Python language and PyVISA package can be downloaded from the web at http://www.python.org/ and http://pyvisa.readthedocs.io/, respectively.

To run this example with Python and PyVISA:

1. Cut-and-paste the code that follows into a file named "example.py".
2. Edit the program to use the VISA address of your oscilloscope.
3. If "python.exe" can be found via your PATH environment variable, open a Command Prompt window; then, change to the folder that contains the "example.py" file, and enter:

```
python example.py
```

```
#!python3
# -*- coding: utf-8 -*-

# ******************************************************
# This program illustrates a few commonly-used programming
# features of your Keysight oscilloscope.
# ******************************************************

# Import modules.
# ----------------------------------------------------------
import pyvisa
import struct
```

```
import sys

# Global variables.
# ------------------------------------------------------------
visa_address = "USB0::0x2A8D::0x4704::Unset::0::INSTR"
input_channel = "CHANnel1"
# input_channel = "CHANnel2"
setup_file_name = "setup.scp"
screen_image_file_name = "screen_image.png"
waveform_data_file_name = "waveform_data.csv"
debug = False


# ============================================================
# Initialize:
# ============================================================
def initialize():

 # Get and display the device's *IDN? string.
 idn_string = do_query_string("*IDN?")
 print(f"Identification string: '{idn_string}'")

 # Clear status and load the default setup.
 do_command("*CLS")
 do_command("*RST")


# ============================================================
# Capture:
# ============================================================
def capture():

 # Use auto-scale to automatically set up oscilloscope.
 print("Autoscale.")
 do_command(":AUToscale")

 # Set trigger mode.
 do_command(":TRIGger:MODE EDGE")
 qresult = do_query_string(":TRIGger:MODE?")
 print(f"Trigger mode: {qresult}")

 # Set EDGE trigger parameters.
 do_command(f":TRIGger:EDGE:SOURce {input_channel}")
 qresult = do_query_string(":TRIGger:EDGE:SOURce?")
 print(f"Trigger edge source: {qresult}")

 do_command(":TRIGger:EDGE:LEVel 1.5")
 # do_command(":TRIGger:EDGE:LEVel -0.008")
 qresult = do_query_string(":TRIGger:EDGE:LEVel?")
 print(f"Trigger edge level: {qresult}")

 do_command(":TRIGger:EDGE:SLOPe POSitive")
 qresult = do_query_string(":TRIGger:EDGE:SLOPe?")
 print(f"Trigger edge slope: {qresult}")

 # Save oscilloscope setup.
 setup_bytes = do_query_ieee_block(":SYSTem:SETup?")
```

```
 f = open(setup_file_name, "wb")
 f.write(setup_bytes)
 f.close()
 print(f"Setup bytes saved: {len(setup_bytes)}")

 # Change oscilloscope settings with individual commands:

 # Set vertical scale and offset.
 do_command(f":{input_channel}:SCALe 0.05")
 # do_command(f":{input_channel}:SCALe 0.5")
 qresult = do_query_string(f":{input_channel}:SCALe?")
 print(f"{input_channel} vertical scale: {qresult}")

 do_command(f":{input_channel}:OFFSet -1.5")
 # do_command(f":{input_channel}:OFFSet -0.008")
 qresult = do_query_string(f":{input_channel}:OFFSet?")
 print(f"{input_channel} offset: {qresult}")

 # Set horizontal scale and offset.
 do_command(":TIMebase:SCALe 0.0002")
 qresult = do_query_string(":TIMebase:SCALe?")
 print(f"Timebase scale: {qresult}")

 do_command(":TIMebase:POSition 0.0")
 qresult = do_query_string(":TIMebase:POSition?")
 print(f"Timebase position: {qresult}")

 # Set the acquisition type.
 do_command(":ACQuire:TYPE NORMal")
 qresult = do_query_string(":ACQuire:TYPE?")
 print(f"Acquire type: {qresult}")

 # Or, set up oscilloscope by loading a previously saved setup.
 setup_bytes = ""
 f = open(setup_file_name, "rb")
 setup_bytes = f.read()
 f.close()
 do_command_ieee_block(":SYSTem:SETup", setup_bytes)
 print(f"Setup bytes restored: {len(setup_bytes)}")

 # Capture an acquisition using :DIGitize.
 do_command(f":DIGitize {input_channel}")


# ============================================================
# Analyze:
# ============================================================
def analyze():

 # Make measurements.
 # ------------------------------------------------------------
 do_command(f":MEASure:SOURce {input_channel}")
 qresult = do_query_string(":MEASure:SOURce?")
 print(f"Measure source: {qresult}")

 do_command(":MEASure:FREQuency")
```

```
qresult = do_query_string(":MEASure:FREQuency?")
print(f"Measured frequency on {input_channel}: {qresult}")

do_command(":MEASure:VAMPlitude")
qresult = do_query_string(":MEASure:VAMPlitude?")
print(f"Measured vertical amplitude on {input_channel}: {qresult}")

# Download the screen image.
# ------------------------------------------------------------
screen_bytes = do_query_ieee_block(":DISPlay:DATA? PNG")

# Save display data values to file.
f = open(screen_image_file_name, "wb")
f.write(screen_bytes)
f.close()
print(f"Screen image written to {screen_image_file_name}.")

# Download waveform data.
# ------------------------------------------------------------

# Set the waveform source.
do_command(f":WAVeform:SOURce {input_channel}")
qresult = do_query_string(":WAVeform:SOURce?")
print(f"Waveform source: {qresult}")

# Set the waveform points mode.
do_command(":WAVeform:POINts:MODE RAW")
qresult = do_query_string(":WAVeform:POINts:MODE?")
print(f"Waveform points mode: {qresult}")

# Get the number of waveform points available.
do_command(":WAVeform:POINts 10240")
qresult = do_query_string(":WAVeform:POINts?")
print(f"Waveform points available: {qresult}")

# Choose the format of the data returned (BYTE or WORD):
do_command(":WAVeform:FORMat WORD")
qresult = do_query_string(":WAVeform:FORMat?")
print(f"Waveform format: {qresult}")

# Specify the byte order in WORD data.
do_command(":WAVeform:BYTeorder LSBF")
qresult = do_query_string(":WAVeform:BYTeorder?")
print(f"Waveform byte order for WORD data: {qresult}")

# Display the waveform settings from preamble:
wav_form_dict = {
 0 : "BYTE",
 1 : "WORD",
 4 : "ASCii",
}
acq_type_dict = {
 0 : "NORMal",
 1 : "PEAK",
 2 : "AVERage",
}
```

```
 preamble_string = do_query_string(":WAVeform:PREamble?")
 (
  wav_form, acq_type, wfmpts, avgcnt, x_increment, x_origin,
  x_reference, y_increment, y_origin, y_reference
 ) = preamble_string.split(",")

 print(f"Waveform format: {wav_form_dict[int(wav_form)]}")
 print(f"Acquire type: {acq_type_dict[int(acq_type)]}")
 print(f"Waveform points desired: {wfmpts}")
 print(f"Waveform average count: {avgcnt}")
 print(f"Waveform X increment: {x_increment}")
 print(f"Waveform X origin: {x_origin}")
 print(f"Waveform X reference: {x_reference}")   # Always 0.
 print(f"Waveform Y increment: {y_increment}")
 print(f"Waveform Y origin: {y_origin}")
 print(f"Waveform Y reference: {y_reference}")

 # Get numeric values for later calculations.
 x_increment = do_query_number(":WAVeform:XINCrement?")
 x_origin = do_query_number(":WAVeform:XORigin?")
 y_increment = do_query_number(":WAVeform:YINCrement?")
 y_origin = do_query_number(":WAVeform:YORigin?")
 y_reference = do_query_number(":WAVeform:YREFerence?")

 # Get the waveform data.
 data_bytes = do_query_ieee_block(":WAVeform:DATA?")
 data_bytes_length = len(data_bytes)
 print(f"Byte count: {data_bytes_length}")

 block_points = data_bytes_length / 2

 # Unpack or split into list of data values.
 values = struct.unpack("%dH" % block_points, data_bytes)
 print(f"Number of data values: {len(values)}")

 # Save waveform data values to CSV file.
 f = open(waveform_data_file_name, "w")

 for i in range(0, len(values) - 1):
  time_val = x_origin + (i * x_increment)
  voltage = ((values[i] - y_reference) * y_increment) + y_origin
  f.write(f"{time_val:E}, {voltage:f}\n")

 # Close output file.
 f.close()
 print(f"Waveform format WORD data written to {waveform_data_file_name}.
")


# ==========================================================
# Send a command and check for errors:
# ==========================================================
def do_command(command, hide_params=False):

 if hide_params:
  (header, data) = command.split(" ", 1)
  if debug:
```

```
   print(f"Cmd = '{header}'")
 else:
  if debug:
   print(f"Cmd = '{command}'")

 InfiniiVision.write(f"{command}")

 if hide_params:
  check_instrument_errors(header)
 else:
  check_instrument_errors(command)


# ============================================================
# Send a command and binary values and check for errors:
# ============================================================
def do_command_ieee_block(command, values):
 if debug:
  print(f"Cmb = '{command}'")
 InfiniiVision.write_binary_values(f"{command} ", values, datatype='B')
 check_instrument_errors(command)


# ============================================================
# Send a query, check for errors, return string:
# ============================================================
def do_query_string(query):
 if debug:
  print(f"Qys = '{query}'")
 result = InfiniiVision.query(f"{query}")
 check_instrument_errors(query)
 return result.strip()


# ============================================================
# Send a query, check for errors, return floating-point value:
# ============================================================
def do_query_number(query):
 if debug:
  print(f"Qyn = '{query}'")
 results = InfiniiVision.query(f"{query}")
 check_instrument_errors(query)
 return float(results)


# ============================================================
# Send a query, check for errors, return binary values:
# ============================================================
def do_query_ieee_block(query):
 if debug:
  print(f"Qyb = '{query}'")
 result = InfiniiVision.query_binary_values(f"{query}", datatype='s', co
ntainer=bytes)
 check_instrument_errors(query)
 return result
```

```
# ========================================================
# Check for instrument errors:
# ========================================================
def check_instrument_errors(command):

 while True:
  error_string = InfiniiVision.query(":SYSTem:ERRor?")
  if error_string:   # If there is an error string value.

   if error_string.find("+0,", 0, 3) == -1:   # Not "No error".

    print(f"ERROR: {error_string}, command: '{command}'")
    print("Exited because of error.")
    sys.exit(1)

   else:   # "No error"
    break

  else:   # :SYSTem:ERRor? should always return string.
   print(f"ERROR: :SYSTem:ERRor? returned nothing, command: '{command}'"
)
   print("Exited because of error.")
   sys.exit(1)


# ========================================================
# Main program:
# ========================================================

rm = pyvisa.ResourceManager("C:\\Windows\\System32\\visa64.dll")
InfiniiVision = rm.open_resource(visa_address)

InfiniiVision.timeout = 30000
InfiniiVision.clear()

# Initialize the oscilloscope, capture data, and analyze.
initialize()
capture()
analyze()

InfiniiVision.close()
print("End of program.")
sys.exit()
```

VISA.NET Examples

These programming examples show how to use the VISA.NET drivers that come with Keysight IO Libraries Suite.

- "VISA.NET Example in C#" on page 1373
- "VISA.NET Example in Visual Basic .NET" on page 1379
- "VISA.NET Example in Python" on page 1385

VISA.NET Example in C#

To compile and run this example in Microsoft Visual Studio 2019:

1. Open Visual Studio.
2. Create a new Visual C#, Windows, Console Application project using the **.NET Framework 4.7.2**.
3. Cut-and-paste the code that follows into the C# source file.
4. Edit the program to use the VISA address of your oscilloscope.
5. Add a reference to the VISA.NET driver:
   a. Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment.
   b. Choose **Add > Reference...**.
   c. In the Reference Manager dialog box, under **Assemblies**, select **Extensions**.
   d. In the "Targeting: .NET Framework 4.7.2" list, select the **Ivi.Visa Assembly** check box; then, click **OK**.
6. Build and run the program.

For more information, see the VISA.NET Help that comes with Keysight IO Libraries Suite.

```
/*
 * Keysight VISA.NET Example in C#
 * ------------------------------------------------------------------
 * This program illustrates a few commonly used programming
 * features of your Keysight InfiniiVision oscilloscope.
 * ------------------------------------------------------------------
 */

using System;
using System.IO;
using System.Collections.Generic;
using System.Text;

using Ivi.Visa;
using Ivi.Visa.FormattedIO;
```

```
namespace Example
{
  class Program
  {

    static void Main(string[] args)
    {
      // Change this variable to the address of your instrument
      string VISA_ADDRESS = "USB0::0x2A8D::0x4704::Unset::0::INSTR";

      // Create a connection (session) to the instrument
      IMessageBasedSession session;
      try
      {
        session = GlobalResourceManager.Open(VISA_ADDRESS) as
          IMessageBasedSession;
      }
      catch (NativeVisaException visaException)
      {
        Console.WriteLine("Couldn't connect.");
        Console.WriteLine("Error is:\r\n{0}\r\n", visaException);
        Console.WriteLine("Press any key to exit...");
        Console.ReadKey();
        return;
      }

      // Create a formatted I/O object which will help us format the
      // data we want to send/receive to/from the instrument
      MessageBasedFormattedIO myScope =
        new MessageBasedFormattedIO(session);

      // For Serial and TCP/IP socket connections enable the read
      // Termination Character, or read's will timeout
      if (session.ResourceName.Contains("ASRL") ||
          session.ResourceName.Contains("SOCKET"))
        session.TerminationCharacterEnabled = true;

      session.TimeoutMilliseconds = 20000;

      // Initialize - start from a known state.
      // ================================================================
      string strResults;
      FileStream fStream;

      // Get and display the device's *IDN? string.
      myScope.WriteLine("*IDN?");
      strResults = myScope.ReadLine();
      Console.WriteLine("*IDN? result is: {0}", strResults);

      // Clear status and load the default setup.
      myScope.WriteLine("*CLS");
      myScope.WriteLine("*RST");

      // Capture data.
      // ================================================================
      // Use auto-scale to automatically configure oscilloscope.
      myScope.WriteLine(":AUToscale");
```

```
// Set trigger mode (EDGE, PULSe, PATTern, etc., and input source.
myScope.WriteLine(":TRIGger:MODE EDGE");
myScope.WriteLine(":TRIGger:MODE?");
strResults = myScope.ReadLine();
Console.WriteLine("Trigger mode: {0}", strResults);

// Set EDGE trigger parameters.
myScope.WriteLine(":TRIGger:EDGE:SOURce CHANnel1");
myScope.WriteLine(":TRIGger:EDGE:SOURce?");
strResults = myScope.ReadLine();
Console.WriteLine("Trigger edge source: {0}", strResults);

myScope.WriteLine(":TRIGger:EDGE:LEVel 1.5");
myScope.WriteLine(":TRIGger:EDGE:LEVel?");
strResults = myScope.ReadLine();
Console.WriteLine("Trigger edge level: {0}", strResults);

myScope.WriteLine(":TRIGger:EDGE:SLOPe POSitive");
myScope.WriteLine(":TRIGger:EDGE:SLOPe?");
strResults = myScope.ReadLine();
Console.WriteLine("Trigger edge slope: {0}", strResults);

// Save oscilloscope configuration.
byte[] ResultsArray;   // Results array.
int nLength;   // Number of bytes returned from instrument.
string strPath;

// Query and read setup string.
myScope.WriteLine(":SYSTem:SETup?");
ResultsArray = myScope.ReadLineBinaryBlockOfByte();
nLength = ResultsArray.Length;

// Write setup string to file.
strPath = "c:\\scope\\config\\setup.scp";
fStream = File.Open(strPath, FileMode.Create);
fStream.Write(ResultsArray, 0, nLength);
fStream.Close();
Console.WriteLine("Setup bytes saved: {0}", nLength);

// Change settings with individual commands:

// Set vertical scale and offset.
myScope.WriteLine(":CHANnel1:SCALe 0.05");
myScope.WriteLine(":CHANnel1:SCALe?");
strResults = myScope.ReadLine();
Console.WriteLine("Channel 1 vertical scale: {0}", strResults);

myScope.WriteLine(":CHANnel1:OFFSet -1.5");
myScope.WriteLine(":CHANnel1:OFFSet?");
strResults = myScope.ReadLine();
Console.WriteLine("Channel 1 vertical offset: {0}", strResults);

// Set horizontal scale and offset.
myScope.WriteLine(":TIMebase:SCALe 0.0002");
myScope.WriteLine(":TIMebase:SCALe?");
strResults = myScope.ReadLine();
```

```
Console.WriteLine("Timebase scale: {0}", strResults);

myScope.WriteLine(":TIMebase:POSition 0.0");
myScope.WriteLine(":TIMebase:POSition?");
strResults = myScope.ReadLine();
Console.WriteLine("Timebase position: {0}", strResults);

// Set the acquisition type (NORMal, PEAK, or AVERage).
myScope.WriteLine(":ACQuire:TYPE NORMal");
myScope.WriteLine(":ACQuire:TYPE?");
strResults = myScope.ReadLine();
Console.WriteLine("Acquire type: {0}", strResults);

// Or, configure by loading a previously saved setup.
byte[] DataArray;
int nBytesWritten;

// Read setup string from file.
strPath = "c:\\scope\\config\\setup.scp";
DataArray = File.ReadAllBytes(strPath);
nBytesWritten = DataArray.Length;

// Restore setup string.
myScope.Write(":SYSTem:SETup ");
myScope.WriteBinary(DataArray);
myScope.WriteLine("");
Console.WriteLine("Setup bytes restored: {0}", nBytesWritten);

// Capture an acquisition using :DIGitize.
myScope.WriteLine(":DIGitize CHANnel1");

// Analyze the captured waveform.
// ============================================================

// Make a couple of measurements.
// ------------------------------------------------------------
myScope.WriteLine(":MEASure:SOURce CHANnel1");
myScope.WriteLine(":MEASure:SOURce?");
strResults = myScope.ReadLine();
Console.WriteLine("Measure source: {0}", strResults);

double fResult;
myScope.WriteLine(":MEASure:FREQuency");
myScope.WriteLine(":MEASure:FREQuency?");
fResult = myScope.ReadLineDouble();
Console.WriteLine("Frequency: {0:F4} kHz", fResult / 1000);

myScope.WriteLine(":MEASure:VAMPlitude");
myScope.WriteLine(":MEASure:VAMPlitude?");
fResult = myScope.ReadLineDouble();
Console.WriteLine("Vertical amplitude: {0:F2} V", fResult);

// Download the screen image.
// ------------------------------------------------------------

// Get the screen data.
myScope.WriteLine(":DISPlay:DATA? PNG");
```

```
ResultsArray = myScope.ReadLineBinaryBlockOfByte();
nLength = ResultsArray.Length;

// Store the screen data to a file.
strPath = "c:\\scope\\data\\screen.png";
fStream = File.Open(strPath, FileMode.Create);
fStream.Write(ResultsArray, 0, nLength);
fStream.Close();
Console.WriteLine("Screen image ({0} bytes) written to {1}",
    nLength, strPath);

// Download waveform data.
// ------------------------------------------------------------

// Set the waveform points mode.
myScope.WriteLine(":WAVeform:POINts:MODE RAW");
myScope.WriteLine(":WAVeform:POINts:MODE?");
strResults = myScope.ReadLine();
Console.WriteLine("Waveform points mode: {0}", strResults);

// Get the number of waveform points available.
myScope.WriteLine(":WAVeform:POINts 10240");
myScope.WriteLine(":WAVeform:POINts?");
strResults = myScope.ReadLine();
Console.WriteLine("Waveform points available: {0}", strResults);

// Set the waveform source.
myScope.WriteLine(":WAVeform:SOURce CHANnel1");
myScope.WriteLine(":WAVeform:SOURce?");
strResults = myScope.ReadLine();
Console.WriteLine("Waveform source: {0}", strResults);

// Choose the format of the data returned (WORD, BYTE, ASCII):
myScope.WriteLine(":WAVeform:FORMat WORD");
myScope.WriteLine(":WAVeform:FORMat?");
strResults = myScope.ReadLine();
Console.WriteLine("Waveform format: {0}", strResults);

// Specify the byte order in the word format data:
myScope.WriteLine(":WAVeform:BYTeorder MSBF");
myScope.WriteLine(":WAVeform:BYTeorder?");
strResults = myScope.ReadLine();
Console.WriteLine("Waveform byte order: {0}", strResults);

// Display the waveform settings:
double[] fResultsArray;
myScope.WriteLine(":WAVeform:PREamble?");
fResultsArray = myScope.ReadLineListOfDouble();

double fFormat = fResultsArray[0];
if (fFormat == 0.0)
{
  Console.WriteLine("Waveform format: BYTE");
}
else if (fFormat == 1.0)
{
  Console.WriteLine("Waveform format: WORD");
```

```
}
else if (fFormat == 2.0)
{
  Console.WriteLine("Waveform format: ASCii");
}

double fType = fResultsArray[1];
if (fType == 0.0)
{
  Console.WriteLine("Acquire type: NORMal");
}
else if (fType == 1.0)
{
  Console.WriteLine("Acquire type: PEAK");
}
else if (fType == 2.0)
{
  Console.WriteLine("Acquire type: AVERage");
}

double fPoints = fResultsArray[2];
Console.WriteLine("Waveform points: {0:e}", fPoints);

double fCount = fResultsArray[3];
Console.WriteLine("Waveform average count: {0:e}", fCount);

double fXincrement = fResultsArray[4];
Console.WriteLine("Waveform X increment: {0:e}", fXincrement);

double fXorigin = fResultsArray[5];
Console.WriteLine("Waveform X origin: {0:e}", fXorigin);

double fXreference = fResultsArray[6];
Console.WriteLine("Waveform X reference: {0:e}", fXreference);

double fYincrement = fResultsArray[7];
Console.WriteLine("Waveform Y increment: {0:e}", fYincrement);

double fYorigin = fResultsArray[8];
Console.WriteLine("Waveform Y origin: {0:e}", fYorigin);

double fYreference = fResultsArray[9];
Console.WriteLine("Waveform Y reference: {0:e}", fYreference);

// Read waveform data.
ushort[] u16ResultsArray;   // Results array.
myScope.WriteLine(":WAVeform:DATA?");
u16ResultsArray = myScope.ReadLineBinaryBlockOfUInt16();
nLength = u16ResultsArray.Length;
Console.WriteLine("Number of data values: {0}", nLength);

// Set up output file:
strPath = "c:\\scope\\data\\waveform_data.csv";
if (File.Exists(strPath)) File.Delete(strPath);

// Open file for output.
StreamWriter writer = File.CreateText(strPath);
```

```
        // Output waveform data in CSV format.
        for (int i = 0; i < nLength - 1; i++)
          writer.WriteLine("{0:f9}, {1:f6}",
              fXorigin + ((float)i * fXincrement),
              (((float)u16ResultsArray[i] - fYreference)
              * fYincrement) + fYorigin);

        // Close output file.
        writer.Close();
        Console.WriteLine("Waveform format WORD data written to {0}",
            strPath);

        // Close the connection to the instrument
        // ------------------------------------------------------------
        session.Dispose();

        Console.WriteLine("Press any key to exit...");
        Console.ReadKey();

      }
    }
}
```

VISA.NET Example in Visual Basic .NET

To compile and run this example in Microsoft Visual Studio 2019:

1. Open Visual Studio.
2. Create a new Visual Basic, Windows, Console Application project using the **.NET Framework 4.7.2**.
3. Cut-and-paste the code that follows into the Visual Basic .NET source file.
4. Edit the program to use the VISA address of your oscilloscope.
5. Add a reference to the VISA.NET driver:
   a. Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment.
   b. Choose **Add > Reference...**.
   c. In the Reference Manager dialog box, under **Assemblies**, select **Extensions**.
   d. In the "Targeting: .NET Framework 4.7.2" list, select the **Ivi.Visa Assembly** check box; then, click **OK**.
6. Specify the Startup object:
   a. Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment.
   b. Choose **Properties**.
   c. In the Properties dialog box, under **Application**, select the **Startup object:** field and choose **Sub Main**.
   d. Save your change and close the Properties dialog box.

7 Build and run the program.

For more information, see the VISA.NET driver help that comes with Keysight Command Expert.

```
'
' Keysight VISA.NET Example in VB.NET
' ----------------------------------------------------------------------
' This program illustrates a few commonly used programming
' features of your Keysight InfiniiVision oscilloscope.
' ----------------------------------------------------------------------

Imports System
Imports System.IO
Imports System.Collections.Generic
Imports System.Text

Imports Ivi.Visa
Imports Ivi.Visa.FormattedIO


Namespace Example
  Class Program

    Public Shared Sub Main(args As String())
      ' Change this variable to the address of your instrument
      Dim VISA_ADDRESS As String = "USB0::0x2A8D::0x4704::Unset::0::INST
R"

      ' Create a connection (session) to the instrument
      Dim session As IMessageBasedSession
      Try
        session = TryCast(GlobalResourceManager.Open(VISA_ADDRESS),
            IMessageBasedSession)
      Catch visaException As NativeVisaException
        Console.WriteLine("Couldn't connect.")
        Console.WriteLine("Error is:" & vbCr & vbLf & "{0}" _
            & vbCr & vbLf, visaException)
        Console.WriteLine("Press any key to exit...")
        Console.ReadKey()
        Return
      End Try

      ' Create a formatted I/O object which will help us format the
      ' data we want to send/receive to/from the instrument
      Dim myScope As New MessageBasedFormattedIO(session)

      ' For Serial and TCP/IP socket connections enable the read
      ' Termination Character, or read's will timeout
      If session.ResourceName.Contains("ASRL") OrElse
        session.ResourceName.Contains("SOCKET") Then
        session.TerminationCharacterEnabled = True
      End If

      session.TimeoutMilliseconds = 20000

      ' Initialize - start from a known state.
```

```
' ================================================================
Dim strResults As String
Dim fStream As FileStream

' Get and display the device's *IDN? string.
myScope.WriteLine("*IDN?")
strResults = myScope.ReadLine()
Console.WriteLine("*IDN? result is: {0}", strResults)

' Clear status and load the default setup.
myScope.WriteLine("*CLS")
myScope.WriteLine("*RST")

' Capture data.
' ================================================================
' Use auto-scale to automatically configure oscilloscope.
myScope.WriteLine(":AUToscale")

' Set trigger mode (EDGE, PULSe, PATTern, etc., and input source.
myScope.WriteLine(":TRIGger:MODE EDGE")
myScope.WriteLine(":TRIGger:MODE?")
strResults = myScope.ReadLine()
Console.WriteLine("Trigger mode: {0}", strResults)

' Set EDGE trigger parameters.
myScope.WriteLine(":TRIGger:EDGE:SOURce CHANnel1")
myScope.WriteLine(":TRIGger:EDGE:SOURce?")
strResults = myScope.ReadLine()
Console.WriteLine("Trigger edge source: {0}", strResults)

myScope.WriteLine(":TRIGger:EDGE:LEVel 1.5")
myScope.WriteLine(":TRIGger:EDGE:LEVel?")
strResults = myScope.ReadLine()
Console.WriteLine("Trigger edge level: {0}", strResults)

myScope.WriteLine(":TRIGger:EDGE:SLOPe POSitive")
myScope.WriteLine(":TRIGger:EDGE:SLOPe?")
strResults = myScope.ReadLine()
Console.WriteLine("Trigger edge slope: {0}", strResults)

' Save oscilloscope configuration.
Dim ResultsArray As Byte()
' Results array.
Dim nLength As Integer
' Number of bytes returned from instrument.
Dim strPath As String

' Query and read setup string.
myScope.WriteLine(":SYSTem:SETup?")
ResultsArray = myScope.ReadLineBinaryBlockOfByte()
nLength = ResultsArray.Length

' Write setup string to file.
strPath = "c:\scope\config\setup.scp"
fStream = File.Open(strPath, FileMode.Create)
fStream.Write(ResultsArray, 0, nLength)
fStream.Close()
```

```
Console.WriteLine("Setup bytes saved: {0}", nLength)

' Change settings with individual commands:

' Set vertical scale and offset.
myScope.WriteLine(":CHANnel1:SCALe 0.05")
myScope.WriteLine(":CHANnel1:SCALe?")
strResults = myScope.ReadLine()
Console.WriteLine("Channel 1 vertical scale: {0}", strResults)

myScope.WriteLine(":CHANnel1:OFFSet -1.5")
myScope.WriteLine(":CHANnel1:OFFSet?")
strResults = myScope.ReadLine()
Console.WriteLine("Channel 1 vertical offset: {0}", strResults)

' Set horizontal scale and offset.
myScope.WriteLine(":TIMebase:SCALe 0.0002")
myScope.WriteLine(":TIMebase:SCALe?")
strResults = myScope.ReadLine()
Console.WriteLine("Timebase scale: {0}", strResults)

myScope.WriteLine(":TIMebase:POSition 0.0")
myScope.WriteLine(":TIMebase:POSition?")
strResults = myScope.ReadLine()
Console.WriteLine("Timebase position: {0}", strResults)

' Set the acquisition type (NORMal, PEAK, or AVERage).
myScope.WriteLine(":ACQuire:TYPE NORMal")
myScope.WriteLine(":ACQuire:TYPE?")
strResults = myScope.ReadLine()
Console.WriteLine("Acquire type: {0}", strResults)

' Or, configure by loading a previously saved setup.
Dim DataArray As Byte()
Dim nBytesWritten As Integer

' Read setup string from file.
strPath = "c:\scope\config\setup.scp"
DataArray = File.ReadAllBytes(strPath)
nBytesWritten = DataArray.Length

' Restore setup string.
myScope.Write(":SYSTem:SETup ")
myScope.WriteBinary(DataArray)
myScope.WriteLine("")
Console.WriteLine("Setup bytes restored: {0}", nBytesWritten)

' Capture an acquisition using :DIGitize.
myScope.WriteLine(":DIGitize CHANnel1")

' Analyze the captured waveform.
' ==========================================================

' Make a couple of measurements.
' ----------------------------------------------------------
myScope.WriteLine(":MEASure:SOURce CHANnel1")
myScope.WriteLine(":MEASure:SOURce?")
```

```
strResults = myScope.ReadLine()
Console.WriteLine("Measure source: {0}", strResults)

Dim fResult As Double
myScope.WriteLine(":MEASure:FREQuency")
myScope.WriteLine(":MEASure:FREQuency?")
fResult = myScope.ReadLineDouble()
Console.WriteLine("Frequency: {0:F4} kHz", fResult / 1000)

myScope.WriteLine(":MEASure:VAMPlitude")
myScope.WriteLine(":MEASure:VAMPlitude?")
fResult = myScope.ReadLineDouble()
Console.WriteLine("Vertical amplitude: {0:F2} V", fResult)

' Download the screen image.
' ------------------------------------------------------------

' Get the screen data.
myScope.WriteLine(":DISPlay:DATA? PNG")
ResultsArray = myScope.ReadLineBinaryBlockOfByte()
nLength = ResultsArray.Length

' Store the screen data to a file.
strPath = "c:\scope\data\screen.png"
fStream = File.Open(strPath, FileMode.Create)
fStream.Write(ResultsArray, 0, nLength)
fStream.Close()
Console.WriteLine("Screen image ({0} bytes) written to {1}",
    nLength, strPath)

' Download waveform data.
' ------------------------------------------------------------

' Set the waveform points mode.
myScope.WriteLine(":WAVeform:POINts:MODE RAW")
myScope.WriteLine(":WAVeform:POINts:MODE?")
strResults = myScope.ReadLine()
Console.WriteLine("Waveform points mode: {0}", strResults)

' Get the number of waveform points available.
myScope.WriteLine(":WAVeform:POINts 10240")
myScope.WriteLine(":WAVeform:POINts?")
strResults = myScope.ReadLine()
Console.WriteLine("Waveform points available: {0}", strResults)

' Set the waveform source.
myScope.WriteLine(":WAVeform:SOURce CHANnel1")
myScope.WriteLine(":WAVeform:SOURce?")
strResults = myScope.ReadLine()
Console.WriteLine("Waveform source: {0}", strResults)

' Choose the format of the data returned (WORD, BYTE, ASCII):
myScope.WriteLine(":WAVeform:FORMat WORD")
myScope.WriteLine(":WAVeform:FORMat?")
strResults = myScope.ReadLine()
Console.WriteLine("Waveform format: {0}", strResults)
```

```
' Specify the byte order in the word format data:
myScope.WriteLine(":WAVeform:BYTeorder MSBF")
myScope.WriteLine(":WAVeform:BYTeorder?")
strResults = myScope.ReadLine()
Console.WriteLine("Waveform byte order: {0}", strResults)

' Display the waveform settings:
Dim fResultsArray As Double()
myScope.WriteLine(":WAVeform:PREamble?")
fResultsArray = myScope.ReadLineListOfDouble()

Dim fFormat As Double = fResultsArray(0)
If fFormat = 0.0 Then
  Console.WriteLine("Waveform format: BYTE")
ElseIf fFormat = 1.0 Then
  Console.WriteLine("Waveform format: WORD")
ElseIf fFormat = 2.0 Then
  Console.WriteLine("Waveform format: ASCii")
End If

Dim fType As Double = fResultsArray(1)
If fType = 0.0 Then
  Console.WriteLine("Acquire type: NORMal")
ElseIf fType = 1.0 Then
  Console.WriteLine("Acquire type: PEAK")
ElseIf fType = 2.0 Then
  Console.WriteLine("Acquire type: AVERage")
End If

Dim fPoints As Double = fResultsArray(2)
Console.WriteLine("Waveform points: {0:e}", fPoints)

Dim fCount As Double = fResultsArray(3)
Console.WriteLine("Waveform average count: {0:e}", fCount)

Dim fXincrement As Double = fResultsArray(4)
Console.WriteLine("Waveform X increment: {0:e}", fXincrement)

Dim fXorigin As Double = fResultsArray(5)
Console.WriteLine("Waveform X origin: {0:e}", fXorigin)

Dim fXreference As Double = fResultsArray(6)
Console.WriteLine("Waveform X reference: {0:e}", fXreference)

Dim fYincrement As Double = fResultsArray(7)
Console.WriteLine("Waveform Y increment: {0:e}", fYincrement)

Dim fYorigin As Double = fResultsArray(8)
Console.WriteLine("Waveform Y origin: {0:e}", fYorigin)

Dim fYreference As Double = fResultsArray(9)
Console.WriteLine("Waveform Y reference: {0:e}", fYreference)

' Read waveform data.
Dim u16ResultsArray As UShort()
myScope.WriteLine(":WAVeform:DATA?")
u16ResultsArray = myScope.ReadLineBinaryBlockOfUInt16()
```

```
        nLength = u16ResultsArray.Length
        Console.WriteLine("Number of data values: {0}", nLength)

        ' Set up output file:
        strPath = "c:\scope\data\waveform_data.csv"
        If File.Exists(strPath) Then
          File.Delete(strPath)
        End If

        ' Open file for output.
        Dim writer As StreamWriter = File.CreateText(strPath)

        ' Output waveform data in CSV format.
        For i As Integer = 0 To nLength - 2
          writer.WriteLine("{0:f9}, {1:f6}",
              fXorigin + (CSng(i) * fXincrement),
              ((CSng(u16 ResultsArray(i)) - fYreference) _
              * fYincrement) + fYorigin)
        Next

        ' Close output file.
        writer.Close()
        Console.WriteLine("Waveform format WORD data written to {0}",
            strPath)

        ' Close the connection to the instrument
        ' ------------------------------------------------------------
        session.Dispose()

        Console.WriteLine("Press any key to exit...")
        Console.ReadKey()

    End Sub
  End Class
End Namespace
```

VISA.NET Example in Python

You can use the Python programming language with the "Python.NET" package (or IronPython) to control Keysight oscilloscopes.

The Python language and "Python.NET" package can be downloaded from the web at http://www.python.org/ and http://pythonnet.github.io/, respectively.

To run this example with Python and "Python.NET":

1. Cut-and-paste the code that follows into a file named "example.py".
2. Edit the program to use the VISA address of your oscilloscope.
3. If "python.exe" can be found via your PATH environment variable, open a Command Prompt window; then, change to the folder that contains the "example.py" file, and enter:

```
python example.py
```

```
#!python3
# -*- coding: utf-8 -*-

# Keysight VISA.NET Example in Python.NET or IronPython
# ******************************************************
# This program illustrates a few commonly-used programming
# features of your Keysight InfiniiVision oscilloscope.
# ******************************************************

# Import Python modules.
# ----------------------------------------------------------
import sys
sys.path.append("C:\\Program Files\\IVI Foundation\\VISA\\Microsoft.NET\
\Framework64\\v2.0.50727\\VISA.NET Shared Components 7.2.0")

# Import .NET modules.
# ----------------------------------------------------------
import clr
clr.AddReference("Ivi.Visa")
from Ivi.Visa import *
from Ivi.Visa.FormattedIO import *
from System import *
from System.IO import *

# Global variables.
# ----------------------------------------------------------
visa_addr = "USB0::0x2A8D::0x4704::Unset::0::INSTR"
io_timeout_ms = 20000
input_channel = "CHANnel1"
# input_channel = "CHANnel2"
setup_file_name = "setup.scp"
screen_image_file_name = "screen_image.png"
waveform_data_file_name = "waveform_data.csv"


# ==========================================================
# Check for instrument errors:
# ==========================================================
def check_instrument_errors(when):

 errors_found = False
 while True:
  # Keep reading errors until "No error".
  myScope.WriteLine(":SYSTem:ERRor?")
  error_string = myScope.ReadLine().strip()
  if error_string:   # If there is an error string value.

   if error_string.find("+0,", 0, 3) == -1:   # Not "No error".
    errors_found = True
    print(f"ERROR: {error_string}")

   else:   # "No error"
    break

  else:   # :SYSTem:ERRor? should always return string.
   errors_found = True
   print("ERROR: ':SYSTem:ERRor?' empty.")
```

```
    break

  if errors_found:
    print(f"Exited because error(s) found when: '{when}'")
    sys.exit(1)


# ============================================================
# Initialize:
# ============================================================
def initialize():

  # Get and display the device's *IDN? string.
  myScope.WriteLine("*IDN?")
  idn_string = myScope.ReadLine().strip()
  print(f"Identification string: '{idn_string}'")

  # Clear status and load the default setup.
  myScope.WriteLine("*CLS")
  myScope.WriteLine("*RST")


# ============================================================
# Capture:
# ============================================================
def capture():

  # Display the input channel.
  myScope.WriteLine(":CHANnel1:DISPlay OFF")
  myScope.WriteLine(f":{input_channel}:DISPlay ON")

  # Use auto-scale to automatically set up oscilloscope.
  print("Autoscale.")
  myScope.WriteLine(":AUToscale:CHANnels DISPlayed")
  myScope.WriteLine(":AUToscale")

  # Set trigger mode.
  myScope.WriteLine(":TRIGger:MODE EDGE")
  myScope.WriteLine(":TRIGger:MODE?")
  qresult = myScope.ReadLine().strip()
  print(f"Trigger mode: {qresult}")

  # Set EDGE trigger parameters.
  myScope.WriteLine(f":TRIGger:EDGE:SOURce {input_channel}")
  myScope.WriteLine(":TRIGger:EDGE:SOURce?")
  qresult = myScope.ReadLine().strip()
  print(f"Trigger edge source: {qresult}")

  myScope.WriteLine(":TRIGger:EDGE:LEVel 1.5")
  myScope.WriteLine(":TRIGger:EDGE:LEVel?")
  qresult = myScope.ReadLine().strip()
  print(f"Trigger edge level: {qresult}")

  myScope.WriteLine(":TRIGger:EDGE:SLOPe POSitive")
  myScope.WriteLine(":TRIGger:EDGE:SLOPe?")
  qresult = myScope.ReadLine().strip()
  print(f"Trigger edge slope: {qresult}")
```

```
# Save oscilloscope setup to file.
myScope.WriteLine(":SYSTem:SETup?")
setup_bytes = myScope.ReadLineBinaryBlockOfByte()
File.WriteAllBytes(setup_file_name, setup_bytes)
print(f"Setup bytes saved: {len(setup_bytes)}")

# Change settings with individual commands:

# Set vertical scale and offset.
myScope.WriteLine(f":{input_channel}:SCALe 0.05")
myScope.WriteLine(f":{input_channel}:SCALe?")
qresult = myScope.ReadLine().strip()
print(f"{input_channel} vertical scale: {qresult}")

myScope.WriteLine(f":{input_channel}:OFFSet -1.5")
myScope.WriteLine(f":{input_channel}:OFFSet?")
qresult = myScope.ReadLine().strip()
print(f"{input_channel} offset: {qresult}")

# Set horizontal scale and offset.
myScope.WriteLine(":TIMebase:SCALe 0.0002")
myScope.WriteLine(":TIMebase:SCALe?")
qresult = myScope.ReadLine().strip()
print(f"Timebase scale: {qresult}")

myScope.WriteLine(":TIMebase:POSition 0.0")
myScope.WriteLine(":TIMebase:POSition?")
qresult = myScope.ReadLine().strip()
print(f"Timebase position: {qresult}")

# Set the acquisition type.
myScope.WriteLine(":ACQuire:TYPE NORMal")
myScope.WriteLine(":ACQuire:TYPE?")
qresult = myScope.ReadLine().strip()
print(f"Acquire type: {qresult}")

# Or, configure by loading a previously saved setup.

# Read setup string from file.
setup_bytes = File.ReadAllBytes(setup_file_name)

# Restore setup string.
myScope.Write(":SYSTem:SETup ")
write_binary = myScope.WriteBinary.Overloads[Array[Byte]]
write_binary(setup_bytes)
myScope.WriteLine("")
print(f"Setup bytes restored: {len(setup_bytes)}")

# Capture an acquisition using :DIGitize.
myScope.WriteLine(f":DIGitize {input_channel}")


# ============================================================
# Analyze:
# ============================================================
def analyze():
```

```
# Make measurements.
# --------------------------------------------------------
myScope.WriteLine(f":MEASure:SOURce {input_channel}")
myScope.WriteLine(":MEASure:SOURce?")
qresult = myScope.ReadLine().strip()
print(f"Measure source: {qresult}")

myScope.WriteLine(":MEASure:FREQuency")
myScope.WriteLine(":MEASure:FREQuency?")
qresult = myScope.ReadLineDouble()
print(f"Measured frequency on {input_channel}: {qresult:f}")

myScope.WriteLine(":MEASure:VAMPlitude")
myScope.WriteLine(":MEASure:VAMPlitude?")
qresult = myScope.ReadLineDouble()
print(f"Measured vertical amplitude on {input_channel}: {qresult:f}")

# Download the screen image.
# --------------------------------------------------------

# Get the screen data.
myScope.WriteLine(":DISPlay:DATA? PNG")
image_bytes = myScope.ReadLineBinaryBlockOfByte()
image_bytes_length = len(image_bytes)
fStream = File.Open(screen_image_file_name, FileMode.Create)
fStream.Write(image_bytes, 0, image_bytes_length)
fStream.Close()
print(f"Screen image written to {screen_image_file_name}.")

# Download waveform data.
# --------------------------------------------------------

# Set the waveform source.
myScope.WriteLine(f":WAVeform:SOURce {input_channel}")
myScope.WriteLine(":WAVeform:SOURce?")
qresult = myScope.ReadLine().strip()
print(f"Waveform source: {qresult}")

# Set the waveform points mode.
myScope.WriteLine(":WAVeform:POINts:MODE RAW")
myScope.WriteLine(":WAVeform:POINts:MODE?")
qresult = myScope.ReadLine().strip()
print(f"Waveform points mode: {qresult}")

# Get the number of waveform points available.
myScope.WriteLine(":WAVeform:POINts 10240")
myScope.WriteLine(":WAVeform:POINts?")
qresult = myScope.ReadLine().strip()
print(f"Waveform points available: {qresult}")

# Choose the format of the data returned (BYTE or WORD):
myScope.WriteLine(":WAVeform:FORMat WORD")
myScope.WriteLine(":WAVeform:FORMat?")
qresult = myScope.ReadLine().strip()
print(f"Waveform format: {qresult}")
```

```
# Display the waveform settings from preamble:
wav_form_dict = {
 0 : "BYTE",
 1 : "WORD",
 4 : "ASCii",
}
acq_type_dict = {
 0 : "NORMal",
 1 : "PEAK",
 2 : "AVERage",
}

myScope.WriteLine(":WAVeform:PREamble?")
(
 wav_form, acq_type, wfmpts, avgcnt, x_increment, x_origin,
 x_reference, y_increment, y_origin, y_reference
) = myScope.ReadLine().split(",")

print(f"Waveform format: {wav_form_dict[int(wav_form)]}")
print(f"Acquire type: {acq_type_dict[int(acq_type)]}")
print(f"Waveform points desired: {wfmpts}")
print(f"Waveform average count: {avgcnt}")
print(f"Waveform X reference: {x_reference}")   # Always 0.

# Get numeric values for later calculations.
myScope.WriteLine(":WAVeform:XINCrement?")
x_increment = myScope.ReadLineDouble()
print(f"Waveform X increment: {x_increment}")
myScope.WriteLine(":WAVeform:XORigin?")
x_origin = myScope.ReadLineDouble()
print(f"Waveform X origin: {x_origin}")
myScope.WriteLine(":WAVeform:YINCrement?")
y_increment = myScope.ReadLineDouble()
print(f"Waveform Y increment: {y_increment}")
myScope.WriteLine(":WAVeform:YORigin?")
y_origin = myScope.ReadLineDouble()
print(f"Waveform Y origin: {y_origin}")
myScope.WriteLine(":WAVeform:YREFerence?")
y_reference = myScope.ReadLineDouble()
print(f"Waveform Y reference: {y_reference}")

# Specify the byte order in WORD data.
myScope.WriteLine(":WAVeform:BYTeorder MSBF")
myScope.WriteLine(":WAVeform:BYTeorder?")
qresult = myScope.ReadLine().strip()
print(f"Waveform byte order for WORD data: {qresult}")

# Get the waveform data values.
myScope.WriteLine(":WAVeform:DATA?")
values = myScope.ReadLineBinaryBlockOfUInt16()
values_length = len(values)
print(f"Number of data values: {values_length}")

# Open file for output.
writer = File.CreateText(waveform_data_file_name)

# Output waveform data in CSV format.
```

```
 for i in range(0, values_length - 1):
  time_val = x_origin + i * x_increment
  voltage = (values[i] - y_reference) * y_increment + y_origin
  writer.WriteLine(f"{time_val:E}, {voltage:f}")

 # Close output file.
 writer.Close()
 print(f"Waveform format WORD data written to {waveform_data_file_name}.
")


# ============================================================
# Main program:
# ============================================================
session = GlobalResourceManager.Open(visa_addr)
session.TimeoutMilliseconds = io_timeout_ms
myScope = MessageBasedFormattedIO(session)

# Initialize the oscilloscope, capture data, and analyze.
initialize()
capture()
analyze()

# Check for oscilloscope instrument errors.
check_instrument_errors("End of program")

# Close the connection to the instrument
session.Dispose()
print("End of program.")

# Wait for a key press before exiting.
print("Press any key to exit...")
Console.ReadKey(True)
```

SICL Examples

SICL Example in C

To compile and run this example in Microsoft Visual Studio 2019:

1. Open Visual Studio.
2. Create a new Visual C++, Windows, Console, Empty project.
3. Cut-and-paste the code that follows into a file named "example.c" in the project directory.
4. In Visual Studio 2019, right-click the Source Files folder, choose **Add > Add Existing Item...**, select the example.c file, and click **Add**.
5. Change the build target from "x86" to "x64".
6. Edit the program to use the VISA address of your oscilloscope.
7. In the Solution Explorer, right-click the project (not the solution), and choose **Properties**. In the Property Pages dialog box, update these project settings:
   a. Under Configuration Properties, Linker, Input, add "sicl32.lib" to the Additional Dependencies field.
   b. Under Configuration Properties, C/C++, Code Generation, select Multi-threaded DLL for the Runtime Library field.
   c. Under Configuration Properties, C/C++, Preprocessor, add "_CRT_SECURE_NO_WARNINGS" to the Preprocessor Definitions.
   d. Under Configuration Properties, VC++ Directories, show directories for **Include Directories**, and add the include directory (for example, Program Files\Keysight\IO Libraries Suite\include).
   e. Under Configuration Properties, VC++ Directories, show directories for **Library Directories**, and add the library files directory (for example, Program Files\Keysight\IO Libraries Suite\lib_x64).
   f. Click **OK** to close the Property Pages dialog.
8. Build and run the program.

```
/*
 * Keysight SICL Example in C
 * ------------------------------------------------------------------
 * This program illustrates a few commonly-used programming
 * features of your Keysight oscilloscope.
 */

#include <stdio.h>            /* For printf(). */
#include <string.h>           /* For strcpy(), strcat(). */
#include <time.h>             /* For clock(). */
#include <sicl.h>             /* Keysight SICL routines. */
```

```
#define SICL_ADDRESS      "usb0[10893::18180::Unset::0]"
#define TIMEOUT           5000
#define IEEEBLOCK_SPACE   3000000

 /* Function prototypes */
void initialize(void);          /* Initialize to known state. */
void capture(void);             /* Capture the waveform. */
void analyze(void);             /* Analyze the captured waveform. */

void do_command(char* command);        /* Send command. */
int do_command_ieeeblock(char* command); /* Command w/IEEE block. */
void do_query_string(char* query);     /* Query for string. */
void do_query_number(char* query);     /* Query for number. */
void do_query_numbers(char* query);    /* Query for numbers. */
int do_query_ieeeblock(char* query);   /* Query for IEEE block. */
void check_instrument_errors();        /* Check for inst errors. */

/* Global variables */
INST id;                        /* Device session ID. */
char str_result[256] = { 0 };   /* Result from do_query_string(). */
double num_result;              /* Result from do_query_number(). */
unsigned char ieeeblock_data[IEEEBLOCK_SPACE];   /* Result from
                                   do_query_ieeeblock(). */
double dbl_results[10];         /* Result from do_query_numbers(). */

/* Main Program
 * ------------------------------------------------------------- */
void main(void)
{
    /* Install a default SICL error handler that logs an error message
     * and exits.  Use the Event Viewer.
     */
    ionerror(I_ERROR_EXIT);

    /* Open a device session using the SICL_ADDRESS */
    id = iopen(SICL_ADDRESS);

    if (id == 0)
    {
        printf("Oscilloscope iopen failed!\n");
    }
    else
    {
        printf("Oscilloscope session opened!\n");
    }

    /* Initialize - start from a known state. */
    initialize();

    /* Capture data. */
    capture();

    /* Analyze the captured waveform. */
    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 the oscilloscope to a known state.
 * ------------------------------------------------------------------- */
void initialize(void)
{
    /* Set the I/O timeout value for this session to 5 seconds. */
    itimeout(id, TIMEOUT);

    /* Clear the interface. */
    iclear(id);

    /* Get and display the device's *IDN? string. */
    do_query_string("*IDN?");
    printf("Oscilloscope *IDN? string: %s\n", str_result);

    /* Clear status and load the default setup. */
    do_command("*CLS");
    do_command("*RST");
}

/* Capture the waveform.
 * ------------------------------------------------------------------- */
void capture(void)
{
    int num_bytes;
    FILE* fp;

    /* Use auto-scale to automatically configure oscilloscope.
     * ------------------------------------------------------------ */
    do_command(":AUToscale");

    /* Set trigger mode (EDGE, PULSe, PATTern, etc., and input source. *
/
    do_command(":TRIGger:MODE EDGE");
    do_query_string(":TRIGger:MODE?");
    printf("Trigger mode: %s\n", str_result);

    /* Set EDGE trigger parameters. */
    do_command(":TRIGger:EDGE:SOURce CHANnel1");
    do_query_string(":TRIGger:EDGE:SOURce?");
    printf("Trigger edge source: %s\n", str_result);

    do_command(":TRIGger:EDGE:LEVel 1.5");
    do_query_string(":TRIGger:EDGE:LEVel?");
    printf("Trigger edge level: %s\n", str_result);

    do_command(":TRIGger:EDGE:SLOPe POSitive");
    do_query_string(":TRIGger:EDGE:SLOPe?");
    printf("Trigger edge slope: %s\n", str_result);
```

```
/* Save oscilloscope configuration.
 * ------------------------------------------------------------ */

 /* Read system setup. */
num_bytes = do_query_ieeeblock(":SYSTem:SETup?");
printf("Read setup string query (%d bytes).\n", num_bytes);

/* Write setup string to file. */
fp = fopen("c:\\scope\\config\\setup.scp", "wb");
num_bytes = fwrite(ieeeblock_data, sizeof(unsigned char), num_bytes,
    fp);
fclose(fp);
printf("Wrote setup string (%d bytes) to ", num_bytes);
printf("c:\\scope\\config\\setup.scp.\n");

/* Change settings with individual commands:
 * ------------------------------------------------------------ */

 /* Set vertical scale and offset. */
do_command(":CHANnel1:SCALe 0.05");
do_query_string(":CHANnel1:SCALe?");
printf("Channel 1 vertical scale: %s\n", str_result);

do_command(":CHANnel1:OFFSet -1.5");
do_query_string(":CHANnel1:OFFSet?");
printf("Channel 1 offset: %s\n", str_result);

/* Set horizontal scale and position. */
do_command(":TIMebase:SCALe 0.0002");
do_query_string(":TIMebase:SCALe?");
printf("Timebase scale: %s\n", str_result);

do_command(":TIMebase:POSition 0.0");
do_query_string(":TIMebase:POSition?");
printf("Timebase position: %s\n", str_result);

/* Set the acquisition type (NORMal, PEAK, or AVERage). */
do_command(":ACQuire:TYPE NORMal");
do_query_string(":ACQuire:TYPE?");
printf("Acquire type: %s\n", str_result);

/* Or, configure by loading a previously saved setup.
* ------------------------------------------------------------ */

/* Read setup string from file. */
fp = fopen("c:\\scope\\config\\setup.scp", "rb");
num_bytes = fread(ieeeblock_data, sizeof(unsigned char),
    IEEEBLOCK_SPACE, fp);
fclose(fp);
printf("Read setup string (%d bytes) from file ", num_bytes);
printf("c:\\scope\\config\\setup.scp.\n");

/* Restore setup string. */
num_bytes = do_command_ieeeblock(":SYSTem:SETup", num_bytes);
printf("Restored setup string (%d bytes).\n", num_bytes);
```

```
    /* Capture an acquisition using :DIGitize.
     * ------------------------------------------------------------- */
    do_command(":DIGitize CHANnel1");
}

/* Analyze the captured waveform.
 * ------------------------------------------------------------------ */
void analyze(void)
{
    double wav_format;
    double acq_type;
    double wav_points;
    double avg_count;
    double x_increment;
    double x_origin;
    double x_reference;
    double y_increment;
    double y_origin;
    double y_reference;

    FILE* fp;
    int num_bytes;   /* Number of bytes returned from instrument. */
    int i;

    /* Make a couple of measurements.
     * ------------------------------------------------------------- */
    do_command(":MEASure:SOURce CHANnel1");
    do_query_string(":MEASure:SOURce?");
    printf("Measure source: %s\n", str_result);

    do_command(":MEASure:FREQuency");
    do_query_number(":MEASure:FREQuency?");
    printf("Frequency: %.4f kHz\n", num_result / 1000);

    do_command(":MEASure:VAMPlitude");
    do_query_number(":MEASure:VAMPlitude?");
    printf("Vertical amplitude: %.2f V\n", num_result);

    /* Download the screen image.
     * ------------------------------------------------------------- */

     /* Read screen image. */
    num_bytes = do_query_ieeeblock(":DISPlay:DATA? PNG");
    printf("Screen image bytes: %d\n", num_bytes);

    /* Write screen image bytes to file. */
    fp = fopen("c:\\scope\\data\\screen.png", "wb");
    num_bytes = fwrite(ieeeblock_data, sizeof(unsigned char), num_bytes,
        fp);
    fclose(fp);
    printf("Wrote screen image (%d bytes) to ", num_bytes);
    printf("c:\\scope\\data\\screen.png.\n");

    /* Download waveform data.
     * ------------------------------------------------------------- */

     /* Set the waveform points mode. */
```

```
do_command(":WAVeform:POINts:MODE RAW");
do_query_string(":WAVeform:POINts:MODE?");
printf("Waveform points mode: %s\n", str_result);

/* Get the number of waveform points available. */
do_command(":WAVeform:POINts 10240");
do_query_string(":WAVeform:POINts?");
printf("Waveform points available: %s\n", str_result);

/* Set the waveform source. */
do_command(":WAVeform:SOURce CHANnel1");
do_query_string(":WAVeform:SOURce?");
printf("Waveform source: %s\n", str_result);

/* Choose the format of the data returned (WORD, BYTE, ASCII): */
do_command(":WAVeform:FORMat WORD");
do_query_string(":WAVeform:FORMat?");
printf("Waveform format: %s\n", str_result);

/* Specify the byte order in the word format data: */
do_command(":WAVeform:BYTeorder LSBF");
do_query_string(":WAVeform:BYTeorder?");
printf("Waveform byte order: %s\n", str_result);

/* Display the waveform settings: */
do_query_numbers(":WAVeform:PREamble?");

wav_format = dbl_results[0];
if (wav_format == 0.0)
{
    printf("Waveform format: BYTE\n");
}
else if (wav_format == 1.0)
{
    printf("Waveform format: WORD\n");
}
else if (wav_format == 2.0)
{
    printf("Waveform format: ASCii\n");
}

acq_type = dbl_results[1];
if (acq_type == 0.0)
{
    printf("Acquire type: NORMal\n");
}
else if (acq_type == 1.0)
{
    printf("Acquire type: PEAK\n");
}
else if (acq_type == 2.0)
{
    printf("Acquire type: AVERage\n");
}

wav_points = dbl_results[2];
printf("Waveform points: %e\n", wav_points);
```

```
    avg_count = dbl_results[3];
    printf("Waveform average count: %e\n", avg_count);

    x_increment = dbl_results[4];
    printf("Waveform X increment: %e\n", x_increment);

    x_origin = dbl_results[5];
    printf("Waveform X origin: %e\n", x_origin);

    x_reference = dbl_results[6];
    printf("Waveform X reference: %e\n", x_reference);

    y_increment = dbl_results[7];
    printf("Waveform Y increment: %e\n", y_increment);

    y_origin = dbl_results[8];
    printf("Waveform Y origin: %e\n", y_origin);

    y_reference = dbl_results[9];
    printf("Waveform Y reference: %e\n", y_reference);

    /* Read waveform data. */
    num_bytes = do_query_ieeeblock(":WAVeform:DATA?");
    printf("Number of data values: %d\n", num_bytes / 2);

    /* Open file for output. */
    fp = fopen("c:\\scope\\data\\waveform_data.csv", "wb");

    /* Output waveform data in CSV format. */
    for (i = 0; i < num_bytes - 1; i+=2)
    {
        /* Write time value, voltage value. */
        fprintf(fp, "%9f, %6f\n",
            x_origin + ((float)(i / 2) * x_increment),
            (((float)(ieeeblock_data[i] + (ieeeblock_data[i + 1] * 256))
            - y_reference) * y_increment) + y_origin);
    }

    /* Close output file. */
    fclose(fp);
    printf("Waveform format WORD data written to ");
    printf("c:\\scope\\data\\waveform_data.csv.\n");
}

/* Send a command to the instrument.
 * ---------------------------------------------------------------- */
void do_command(command)
char* command;
{
    char message[80];

    strcpy(message, command);
    strcat(message, "\n");
    iprintf(id, message);

    check_instrument_errors();
```

```
}

/* Command with IEEE definite-length block.
 * ------------------------------------------------------------------- */
int do_command_ieeeblock(command, num_bytes)
char* command;
int num_bytes;
{
    char message[80];
    int data_length;

    strcpy(message, command);
    strcat(message, " #8%08d");
    iprintf(id, message, num_bytes);
    ifwrite(id, ieeeblock_data, num_bytes, 1, &data_length);

    check_instrument_errors();

    return(data_length);
}

/* Query for a string result.
 * ------------------------------------------------------------------- */
void do_query_string(query)
char* query;
{
    char message[80];

    strcpy(message, query);
    strcat(message, "\n");
    iprintf(id, message);

    iscanf(id, "%t\n", str_result);

    check_instrument_errors();
}

/* Query for a number result.
 * ------------------------------------------------------------------- */
void do_query_number(query)
char* query;
{
    char message[80];

    strcpy(message, query);
    strcat(message, "\n");
    iprintf(id, message);

    iscanf(id, "%lf", &num_result);

    check_instrument_errors();
}

/* Query for numbers result.
 * ------------------------------------------------------------------- */
void do_query_numbers(query)
char* query;
```

```
{
    char message[80];

    strcpy(message, query);
    strcat(message, "\n");
    iprintf(id, message);

    iscanf(id, "%,10lf\n", dbl_results);

    check_instrument_errors();
}

/* Query for an IEEE definite-length block result.
 * ------------------------------------------------------------- */
int do_query_ieeeblock(query)
char* query;
{
    char message[80];
    int data_length;

    strcpy(message, query);
    strcat(message, "\n");
    iprintf(id, message);

    data_length = IEEEBLOCK_SPACE;
    iscanf(id, "%#b", &data_length, ieeeblock_data);

    if (data_length == IEEEBLOCK_SPACE)
    {
        printf("IEEE block buffer full: ");
        printf("May not have received all data.\n");
    }

    check_instrument_errors();

    return(data_length);
}

/* Check for instrument errors.
 * ------------------------------------------------------------- */
void check_instrument_errors()
{
    char str_err_val[256] = { 0 };
    char str_out[800] = "";

    ipromptf(id, ":SYSTem:ERRor?\n", "%t", str_err_val);
    while (strncmp(str_err_val, "+0,No error", 3) != 0)
    {
        strcat(str_out, ", ");
        strcat(str_out, str_err_val);
        ipromptf(id, ":SYSTem:ERRor?\n", "%t", str_err_val);
    }

    if (strcmp(str_out, "") != 0)
    {
        printf("INST Error%s\n", str_out);
        iflush(id, I_BUF_READ | I_BUF_WRITE);
```

```
        }
}
```

SCPI.NET Examples

You can also program the oscilloscope using the SCPI.NET drivers that come with Keysight's free Command Expert software.

While you can write code manually using the SCPI.NET drivers, you can also use the Command Expert software to:

- Connect to instruments and control them interactively using SCPI command sets.
- Quickly prototype and test command sequences.
- Generate C#, VB.NET, or C/C++ code for command sequences.
- Find, download, and install SCPI command sets.
- Browse command trees, search for commands, and view command descriptions.

The Command Expert suite also comes with Add-ons for easy instrument control and measurement data retrieval in NI LabVIEW, Microsoft Excel, Keysight VEE, and Keysight SystemVue.

To download the Keysight Command Expert software, see:
http://www.keysight.com/find/commandexpert

For more on programming with the SCPI.NET drivers, see "Using SCPI.NET Drivers" in the help that comes with Keysight Command Expert.

Index

Symbols

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z